UNIVERZITA KARLOVA V PRAZE

Přírodovědecká fakulta

Katedra aplikované geoinformatiky a kartografie

Studijní program: Geografie (navazující magisterské studium)

Studijní obor: Kartografie a geoinformatika

Bc. Jana SVOBODOVÁ

ALGORITMUS PRO AUTOMATIZOVANOU KARTOGRAFICKOU GENERALIZACI SHLUKŮ BUDOV METODOU AGREGACE

ALGORITHM FOR AUTOMATED BUILDING SIMPLIFICATION USING AGGREGATION

Diplomová práce

Vedoucí diplomové práce: Ing. Tomáš Bayer, Ph.D.

Praha 2012

Prohlašuji, že jsem závěrečnou práci zpracovala samostatně a že jsem uvedla všechny použité informační zdroje a literaturu. Tato práce ani její podstatná část nebyla předložena k získání jiného nebo stejného akademického titulu.

V Brně dne 15. 8. 2012 .....

Jana Svobodová

Na tomto místě bych chtěla poděkovat především vedoucímu diplomové práce Ing. Tomáši Bayerovi, Ph.D. za cenné rady, připomínky a čas věnovaný konzultacím. Dále děkuji rodičům za podporu v průběhu celého studia.

Algoritmus pro automatizovanou kartografickou generalizaci shluků budov metodou agregace

Abstrakt

Diplomová práce se věnuje tématu automatizované kartografické generalizace. Jejím hlavním cílem je navržení nového generalizačního algoritmu pro agregaci budov.

V první části je provedena rešerše algoritmů dosud navržených pro agregaci budov. Dále je podán výklad vlastního návrhu algoritmu - nejprve jsou podrobně popsány pomocné datové struktury a algoritmy, které algoritmus využívá, dále jsou stanoveny kartografické a geometrické podmínky pro agregaci.

Vlastní algoritmus je založen na principu konstrukce straight skeletonu. Ze straight skeletonu jsou odebrány vnější vrcholy, nad takto vzniklými strukturami je provedena agregace a agregovaný polygon je získán zpětnou rekonstrukcí ze své kostry.

Druhá část práce je zaměřena na implementaci a zhodnocení výsledků. Algoritmus je implemetován pomocí open-source knihoven *CGAL*, *Boost* a *Shapelib*. Dosažené výsledky a jejich stručné porovnání se SW ArcGIS je diskutováno v závěru práce.

Klíčová slova: automatizovaná kartografická generalizace, aggregace budov, straight skeleton

Algorithm for automated building simplification using aggregation

Abstract

Diploma thesis deals with automated cartographic generalization. The main aim is to propose a new generalization algorithm for building aggregation.

The first part brings summary of existing algorithms for building aggregation. Then the new algorithm is presented: at first, auxiliary data structures and algorithms are presented, then cartographic and geometric requirements are defined.

New algorithm is based on the principle of straight skeleton construction. Outer vertices are removed from constructed straight skeletons and those structures are aggregated. The aggregated polygon is reconstructed from aggregated structures.

The second part is focused on implementation and results evaluation. The algorithm is implemented using open-source libraries *CGAL*, *Boost* and *Shapelib*. The results and confrontation with SW ArcGIS are discussed in conclusion of the thesis.

Key words: automated cartographic generalization, building aggregation, straight skeleton

Obsah

Seznam obrázků

1 ÚVOD

Kartografická generalizace je v současné době stále živým tématem kartografického výzkumu. Podněty k dalšímu zkoumání a zefektivňování kartografické generalizace jsou dávány jak rozvojem digitálních technologií, tak i nových technik získávání a zpracování prostorových dat (např. laser scanning). Kartografové se této problematice věnovali odjakživa, dříve byla zaměřena především na tištěné mapy, s rozvojem počítačové kartografie se stále více uplatňují automatizované systémy pro generalizaci. První pokusy o návrh algoritmů pro automatizovanou kartografickou generalizaci se objevují již v 60. letech 20. století, nicméně důkladnější zkoumání, společně s rozvojem počítačů, se váže spíše až k počátku 70. let. Jako první se začaly zkoumat algoritmy na generalizaci polylinií, což je relativně snadnější úkol než např. generalizace polygonů. Později se pozornost přesunula i do dalších oblastí, bylo vytvořeno množství koncepčních modelů a nových přístupů k automatizované generalizaci.

Generalizace budov je řešena nejen národními mapovacími agenturami při tvorbě státních mapových děl, ale i v soukromém sektoru. Generalizace může být prováděna buď ručně, nebo poloautomatizovaně.

Cílem kartografů je dospět do stavu existujícího plně automatizovaného systému. Cesty k dosažení takového stavu jsou dvojí: buď se navrhují nové algoritmy, nebo se optimalizují algoritmy stávající. Některé oblasti automatizované kartografické generalizace jsou již poměrně bohatě rozpracovány - např. problematika zjednodušování tvarů budov. Jiné oblasti, mezi něž patří i problematika agregace, jsou rozpracovány méně. Cílem předkládané diplomové práce je tedy navrhnout nový algoritmus, který řeší problematiku kartografické generalizace budov metodou agregace.

Nově navrhovaný algoritmus pracuje ve vektorovém prostředí. Jeho stěžejní myšlenkou je využití topologické kostry polygonu, konkrétně straight skeletonu. Při práci s kostrou polygonu jsou využívány grafové algoritmy, především prohledávání do hloubky a nalezení minimální kostry grafu. Z kostry agregovaného polygonu je rozšiřováním po bisektorech zrekonstruován vlastní výsledný polygon.

Navrhovaný algoritmus se snaží vytvořit alternativu k již existujícím řešením, která více či méně úspěšně řeší problematiku agregace. Hlavní přínos nového algoritmu vidím především v následujících oblastech:

- v pokusu zahrnout agregaci do celkového kontextu kartografické generalizace,
- ve snaze zjednodušit nově vznikající polygon v závislosti na zadaných parametrech (měřítku),
- ve snaze zpracovat co nejširší spektrum budov,
- v návrhu pouze ve vektorovém prostředí.

Konkrétní požadavky na algoritmus jsou dále stanoveny a rozvedeny v 5. kapitole, 6. kapitola se věnuje návrhu algoritmu a 8. kapitola je zaměřena na jeho celkové zhodnocení.

Diplomová práce je rozdělena do dvou hlavních částí - teoretické a praktické. Teoretická část obsahuje šest kapitol - úvod, literární rešerši, shrnutí poznatků v oblasti kartografické generalizace se zaměřením na agregaci a tři kapitoly věnující se návrhu nového algoritmu. Praktická část zahrnuje tři části - implementaci navrženého algoritmu, zhodnocení výsledků a efektivity vytvořeného nástroje a závěr. Práce je uzavřena seznamem použitých zdrojů.

2 REŠERŠE LITERATURY

Rešeršní část diplomové práce se věnuje především generalizačním algoritmům v kartografii se zaměřením na automatické řešení agregace a také je zmíněno několik publikací pojednávajících obecně o (automatizované) kartografické generalizaci.

Ačkoliv je problematika kartografické generalizace řešena souvisle již téměř tři desetiletí, obsáhlejších publikací zabývajících se touto problematikou je pouze několik (existuje však poměrně rozsáhlé množství publikací kratšího rozsahu). Problematika automatizované kartografické generalizace metodou agregace nebyla doposud v kartografické literatuře detailněji řešena, existuje k ní pouze velmi málo relevantních publikací.

V rámci rešerše byly prostudovány dvě knižní publikace a několik článků a závěrečných prací týkajících se problematiky kartografické generalizace, automatizované generalizace budov a automatizované generalizace metodou agregace. Převážná většina literatury je cizojazyčná, zejména anglická, česká literatura je zmíněna na závěr kapitoly.

Dobrý výchozí základ pro pochopení vývoje a problematiky kartografické generalizace nejen teoreticky, ale i s kapitolami o provedených výzkumech, představuje publikace *Generalisation of Geograhic Information: Cartograhic Modelling and Applications* (MACKANESS, RUAS, SARJAKOSKI, 2007). Tato publikace mimo jiné shrnuje dosavadní teoretické přístupy ke kartografické generalizaci.

Publikace *Algorithmic Foundation of Multi-Scale Spatial Representation* (LI, 2007) se zaměřuje především na výklad principu algoritmů používaných v digitální kartografii. V první části je srozumitelně podán výklad rámce pro tyto algoritmy a také matematicko-geometrické základy (matematická morfologie, Delaunay triangulace, Voronoi diagramy, skeletonizace). Druhá část publikace se pak zaměřuje na jednotlivé algoritmy, klasifikované podle generalizačních operátorů a podle dimenzionality prvků (bod, linie, polygon, 2,5-D). V této části jsou vyloženy i základní generalizační algoritmy pro agregaci.

Tyto dvě knihy se staly výchozím materiálem pro studium, dále byly prostudovány články pro hlubší pochopení problematiky.

Jedním z prvních, kdo se zabýval generalizací sídel, byl německý kartograf W. Staufenbiel (1973, Universität Hannover). Navrhl komplexní algoritmus pro generalizaci sídel, včetně vodstva, komunikací, budov a vegetačního pokryvu. V otázce generalizace budov se zaměřuje především na zjednodušení tvaru budov, založenému na odstranění nebo upravení krátkých

stran. Otázce agregace se věnuje jen v náznacích a nenavrhuje žádný konkrétní algoritmus.

Další německý kartograf, U. Meyer (1989), hodnotil a testoval Staufenbielův algoritmus a sám se pokoušel navrhnout algoritmy pro automatické rozeznání zvláštně tvarovaných budov, které Staufenbielův algoritmus nedokáže dobře řešit (podle ŠTAMAPACH, 2006).

Konkrétní návrh algoritmu týkajícího se agregace polygonů vytvořili SU et al, 1997. Tento algoritmus je založen na matematické morfologii a je prováděn v rastrovém prostředí.

V letech 1997-2000 byl pod hlavičkou Evropské komise veden projekt AGENT (Automated GEneralisation New Technology) za účelem vyvinutí nových generalizačních technik. Účastníky projektu byli francouzská NMA (IGN), firma LaserScan a univerzity v Curychu, Edinburgu a Grenoblu. Výsledkem je model hierarchicky pracující, založený na agentech ve třech úrovních (mikro-, meso- a makro- měřítko), a tím je usměrňována generalizace v širokém kontextu. O projektu bylo publikováno množství článků, např. AGENT, DD2, 2001 (přehled základních algoritmů), nebo LAMY et al. (princip fungování).

Zpráva AGENT, DD3, 2001 uvádí přehled algoritmů pro některé operace (mezi nimi i agregaci) a doporučení pro použití těchto algoritmů v rámci generalizace. Z algoritmů pro agregaci jmenujme algoritmy WARE et al., 1995. Jeden je založený na Delaunay triangulaci a následné rotaci a posunutí objektů, druhý také využívá triangulaci a následné vyplnění prostorů mezi objekty. REGNAULD, 1998, navrhl algoritmus založený na posunutí objektů a v témže roce i algoritmus založený na konvexní obálce. ORMSBY, 1999 navrhl podobně jako SU algoritmus založený na matematické morfologii.

Další zpráva AGENT, DA2, 2001, definuje omezující podmínky (constraints) pro generalizaci jednotlivých témat (včetně zástavby) a to na všech třech úrovních (mikro, mezo, makro). Některé z těchto omezení využívají STEINIGER a TAILLANDIER, 2005 při generalizaci budov se snahou o zlepšení této generalizace. Přidávají pravidla pro rozdělení zástavby do pěti typů. Praktický příklad ukázali na generalizaci z měřítka 1:10 000 do měřítka 1:25 000 na švýcarských a francouzských mapách.

Projekt AGENT poskytuje dobré výsledky zejména v urbánních oblastech. Vylepšení přináší model CartACom (DUCHENE, 2004), který se zaměřuje na slabosti předchozího modelu v rurálních oblastech.

V roce 2007 byly tyto dva modely doplněny ještě dalším, který lépe pracuje pro podkladová témata, která souvisle pokrývají plochu (např. landcover, vrstevnice). Tento model se nazývá GAEL a byl vytvořen Gaffurim. Možné způsoby kombinace těchto tří modelů navrhují DUCHENE a GAFFURI, 2008. Otázce agregace se věnuje ve své disertační práci HAUNERT, 2008. Zaměřuje se na optimizaci agregace plošných prvků (landcover, administrativní členění...). Jeho řešení přináší lepší výsledky zejména v otázce sémantické přesnosti, jedná se o deterministickou metodu a umožňuje lepší zpracování omezujících podmínek (constraints).

QI a LI, 2008 v úvodní části svého článku provádí krátké shrnutí dosavadních algoritmů pro generalizaci budov, např. Ruas, 1998, která se zabývala metodou odsunu budov, Regnauld 2001, který zkoumal typifikaci budov nebo Christophe a Ruas, 2002 (detekce linie budov (building alignment)). Oni sami se věnují otázce shlukování budov založenému na hierarchických omezeních, jakými jsou například blízkost, podobnost nebo orientace.

Kartografickou generalizací, zejména její algoritmizací, se také dlouhodobě zabývá BAYER. Příkladem jeho práce je publikace z roku 2009, kde navrhuje algoritmus na zjednodušení budov s využitím rekurze. Tento algoritmus se snaží zlepšovat výsledky především samostatně stojících pravoúhlých budov komplexních geometrických tvarů.

Z česky psaných publikací se problému generalizace budov věnuje ve své diplomové práci ŠTAMPACH, 2006, který implementuje Staufenbielův algoritmus. Testuje jeho úspěšnost na případu města Brna. Navíc také podává stručný přehled algoritmů týkajících se generalizace budov (zjednodušování tvaru, posuny objektů, typifikace).

Diplomová práce POPELÍNSKÉHO, 2011 se věnuje generalizaci říčních sítí, implementuje a porovnává výsledek generalizace na základě různých klasifikací říčních toků a dalších parametrů. Nemá tedy přímou souvislost s generalizací budov, v úvodním přehledu je však provedena krátká rešerše přístupů ke generalizaci.

Uvedený přehled není ani nemůže být vyčerpávající. Většina publikací týkajících se kartografické generalizace je ve formě článků a různých příspěvků v odborných časopisech a na konferencích. Jsou navrhovány nové algoritmy pro různé generalizační operátory, nicméně problematika agregace není příliš často v zájmu výzkumníků.

3 KARTOGRAFICKÁ GENERALIZACE SE ZAMĚŘENÍM NA AGREGACI

V této části je podrobněji vysvětlena problematika kartografické generalizace a přístupy k jejímu řešení, je definována a popsána operace agregace a kapitolu uzavírá přehled existujících algoritmů provádějících agregaci.

3.1 Koncepční modely generalizace

V průběhu času bylo kartografy vytvořeno velké množství koncepčních modelů generalizace. Koncepční model je teoretický podklad pro řešení dané problematiky. Nastiňuje pohled autorů na věc a navrhuje způsob řešení nebo zpracování problému.

Jedním z prvních byl model Robinsona et al. (1978), kteří rozdělili proces generalizace na dva kroky: selekce (předzpracování) a samotná generalizace zahrnující geometrickou a statistickou manipulaci s objekty, která je dosažena skrz operátory simplifikace, klasifikace a symbolizace.

I v dnešní době je často citovaný model McMASTER a SHEA, 1989. Stanovuje tři důležité aspekty: proč generalizovat, kdy generalizovat a jak generalizovat. První část představuje teoretické otázky (stanovení cílů generalizace), druhá část analýzu prostředků, které jsou k dispozici (kartometrické analýzy a dostupné algoritmy) a třetí generalizační operátory, pomocí kterých je generalizace prováděna. Tento model slouží ke zhodnocení, zda je generalizace nutná a pokud ano, tak jakými prostředky v dané situaci jí nejlépe dosáhnout.

Tyto modely jsou stále používané, i když jsou kritizované z důvodu, že přistupují ke generalizaci sekvenčně a nikoliv komplexně. Sekvenční přístup je charakteristický tím, že jednotlivé kroky generalizace jsou řešeny postupně, např. postupnou aplikací jednotlivých generalizačních operátorů na jednotlivé složky databáze (body, linie, polygony; vodstvo, terén, komunikace, zástavba...). Problém nastává v tom, že jednotlivé prvky se navzájem ovlivňují a v toku generalizace je nelze editovat samostatně. Komplexnější řešení se tak snaží navrhnout postup, který zohledňuje právě tyto vazby. Byly proto vytvořeny i další modely, založené na znalostních systémech a umělé inteligenci (např. Muller, 1990, 1991, Buttenfield a McMaster, 1991), modely založené na komunikujících agentech (projekt AGENT) nebo modely s omezeními (constraint-based modelling) - např. Sarjakoski a Kilpeläinen, 1999 nebo Sester, 2005.

Modely se různě vyvíjejí, ale v pozadí stále zůstává geometrický základ jednotlivých operátorů, které generalizaci charakterizují.

3.2 Generalizační operátory

Generalizační operátory reprezentují způsob transformace prvků mapy v průběhu generalizace, resp. transformace, ke kterým v důsledku generalizace dochází. Těchto změn je pak v průběhu generalizace prakticky dosaženo implementací vhodných algoritmů.

Operátory mohou být klasifikovány různými způsoby. Podle LI, 2007 provedli Rieger a Coulson v roce 1993 průzkum mezi uznávanými kartografy, a mnozí z nich se neshodli na definici jednotlivých operátorů nebo některé ani definovat nedokázali. Neexistuje konsensus pro klasifikaci nebo definici operátorů. Zde budeme vycházet z klasifikace použité LI, 2007, která rozděluje operátory podle dimenzionality prvků, na kterých jsou generalizační algoritmy prováděny.

Náš zájem leží především na generalizaci polygonů, konkrétně budov. Bylo vytvořeno množství algoritmů pro operátory pracující jak s jednotlivými polygony, tak i se skupinou polygonů. Téma práce je zaměřeno na problematiku agregace, která dosud není uspokojivě řešena, a proto se v následující části budeme věnovat algoritmům zaměřeným na agregaci a amalgamaci. S operátory agregace a amalgamace bude v této části stručně porovnán i operátor typifikace a proto uvedeme také jeho definici. Definice operátorů jsou následující.

agregace: spojení více prvků oddělených prázdným prostorem v jeden zjednodušený

amalgamace: spojení více prvků oddělených jiným prvkem v jeden zjednodušený

typifikace: zachování typického vzoru více prvků se současným zjednodušením struktury, výsledkem je opět více prvků

Operace	Velké měřítko	Zmenšení	Malé měřítko
Agregace			
Amalgamace			
Typifikace			

Obr. 3.1: Vybrané generalizační operátory (upraveno podle: LI, 2007, s. 23)

LI v klasifikaci tyto operátory odděluje, nicméně v části věnované algoritmům jsou agregace a amalgamace vykládány společně. Tento princip je použit i v diplomové práci.

Závislost na měřítku mapy. Generalizační operátor, který rozsahem své použitelnosti a vhodnosti použití nejvíce soupeří s agregací, je typifikace. Podle LI, 2007, se mění typ použitých operátorů při generalizaci v závislosti na měřítku. Při generalizaci z měřítka 1:10 000 do 1:25 000 je hlavní operací typifikace. Při transformaci do 1:50 000 už se začíná více uplatňovat agregace a amalgamace. Při dalším oddálení (1:100 000) už jsou převažujícími operacemi právě tyto dvě.

Ve zprávě AGENT (DD3, 2001) můžeme nalézt jiný příklad. Zde je agregace pojata jako souhrný název pro amalgamaci a typifikaci. Agregace v našem pojetí je zde nazývána amalgamací, resp. amalgamací, kdy se jedná o sloučení více prvků v jeden větší stejné vrstvy. Cílem této operace je pak zachovat co nejvíce půdorys, plochu i strukturu agregovaných prvků.

Zachování struktury je i cílem typifikace. Kritériem, kdy vybrat mezi agregací a typifikací, je pak geografický kontext. Ve zprávě je uveden příklad rozčlenění města do tří typů zástavby, resp. hustoty zástavby: centrum města, okrajové oblasti a přechodné oblasti. V centru je zástavba znázorněna plnými bloky budov, v přechodné oblasti se uplatňuje agregace pro spojení jednotlivých budov v komplexnější budovy a v okrajových částech se uplatňuje hlavně typifikace pro zachování rozvolněné zástavby.

České mapové sady. Tyto skutečnosti jsou platné i pro české sady mapových děl. Prostudováním Základních a Topografických map ČR různých měřítek (od 1:10 000 do 1:100 000) můžeme zhodnotit znázornění budov jako samostatně stojících objektů následovně:

- v měřítku 1:10 000 jsou znázorněny téměř všechny budovy (při porovnání ZABAGED s ortofoto), i když už tvarově zjednodušené,
- v měřítku 1:25 000 (topografická mapa) dochází k částečné agregaci budov, převažují však spíše jiné operátory (především typifikace a to především v místech suburbánního a rurálního charakteru),
- v měřítku1:50 000 (Základní i topografická mapa) jsou téměř bez výjimky hustě zastavěná území (urbánní oblasti) znázorněna jako bloky budov, místa s nižší hustotou staveb jsou zakreslena zjednodušenými budovami,
- v měřítku 1:100 000 jsou sídla zobrazena buď plošným nebo bodovým znakem v závislosti na jejich velikosti a významu.

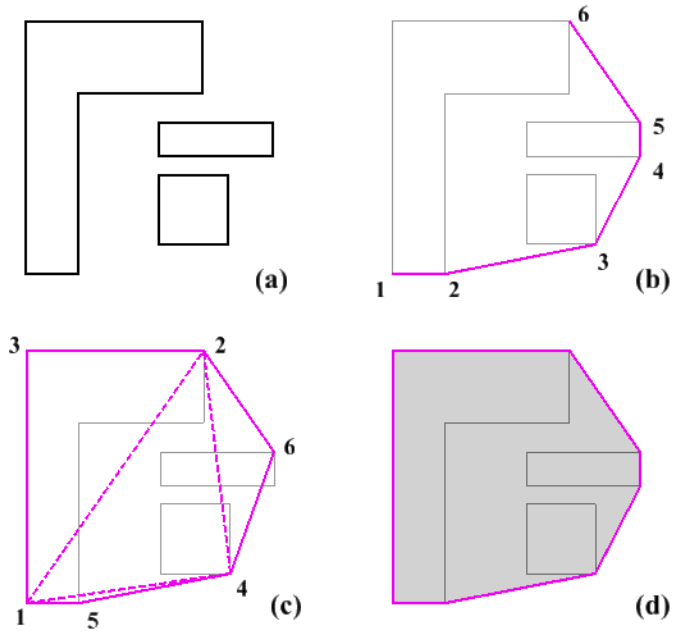
3.3 Algoritmy pro generalizaci skupin polygonů metodami agregace a amalgamace

Algoritmy pro agregaci a amalgamaci můžeme podle LI, 2007 rozdělit na dva typy: algoritmy pracující s hranicí polygonu (*boundary-based*), nebo pracující s oblastí jako celkem (*region-based*). Ve zprávě AGENT, DD3, 2001 jsou algoritmy pro agregaci (resp. amalgamaci) rozděleny podle techniky, kterou je agregace provedena. První skupina pracuje s posunutím prvků (*displacement*), druhá s vyplněním místa mezi objekty. K prvnímu typu algoritmů (dle LI) patří především algoritmy využívající konvexní obálky.

Mezi druhý typ algoritmů můžeme zařadit algoritmus SU et al., 1997, který využívá morfologických operátorů. Ten byl navržen pro rastrové prostředí, kdežto předchozí pracují s vektorovými daty. Nicméně LI, 2007, se domnívá, že není příliš nutné rozlišovat mezi prostředím algoritmů. Algoritmy jsou velmi často aplikované tak, že se vychází z vektoru, který je převeden na rastr, ve kterém se dají snáze definovat prostorové vazby, a jsou na něm aplikovány jednotlivé generalizační kroky. Výsledek generalizace je pak konvertován na vektorová data. Tyto transformace jsou však výpočetně velmi neefektivní a informačně ztrátové.

3.3.1 Boundary-based algoritmy

V nejjednodušší formě je proces agregace pomocí konvexních obálek vlastně jen vytvořením příslušné konvexní obálky. Algoritmů na tvorbu konvexních obálek je velké množství, zmiňme například *gift-wrapping algoritmus* (Jarvis, 1973) a *quick hull algoritmus* (Gosper, 1998; O'Rourke, 1993) nebo *Graham's scan* (Graham, 1972). Výklad principu jednotlivých algoritmů lze nalézt například v LI, 2007, jejich ilustrace je na obr. 3.2.



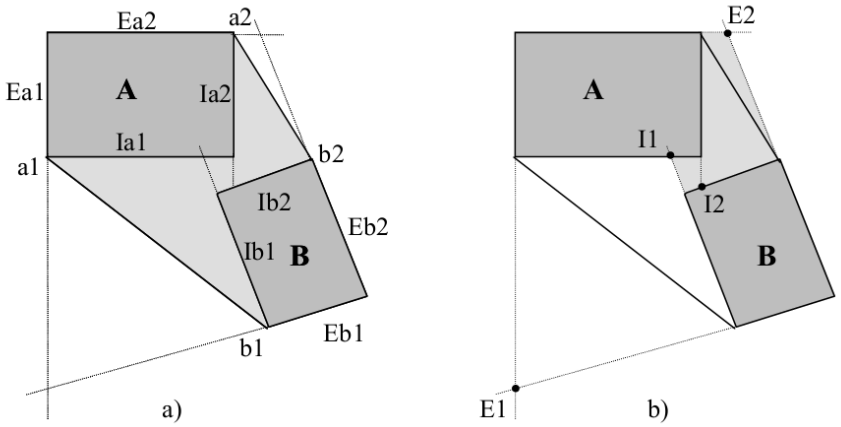
Obr. 3.2: *(a)* budovy vstupující do algoritmu pro konvexní obálku, *(b)* gift-wrapping algoritmus, *(c)* quick hull algoritmus, *(d)* výsledná konvexní obálka

LI dále uvádí, že řešení agregace tímto způsobem je v některých případech velmi zjednodušené. Uvádí možnost přidání nějakých omezení pro tvorbu konvexní obálky (např. maximální délka strany), ale ani toto řešení nepřináší příliš uspokojivé výsledky.

Jak je zřejmé z obrázku 3.2, tyto algoritmy mají tu nevýhodu, že nezachovávají pravé úhly, takže nejsou příliš použitelné pro budovy. To je důsledkem definice konvexní obálky. Obálka vytváří polygon o nejmenší ploše, který obsahuje všechny body útvaru, ale jejími vertexy jsou pouze body původních prvků. Tím nelze přesněji vystihnout tvar agregovaných prvků. Zároveň se také zvětšuje plocha agregovaných oblastí. Dalším problémem může být vznik *self-intersections*.

Existuje také redefinice konvexní obálky zvaná *rectangular convex hull*, která dokáže zachovat pravé úhly. Stále však zůstává problém vzniku topologicky nekorektních polygonů.

Zpráva AGENT, DD3 uvádí alogritmus REGNAULDA, 1998, který také pracuje s konvexní obálkou. Algoritmus funguje vždy pro dvojici objektů, v případě nutnosti spojení více prvků se agregují postupně. Princip algoritmu uvádí obr. 3.3.



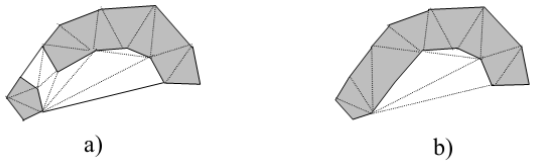
Obr. 3.3: Agregace konvexní obálkou, REGNAULD, 1998 (převzato z: AGENT, DD3, 2001, s. 39)

Na úvod algoritmus vytváří konvexní obálku, která je dále modifikována tak, aby lépe vystihovala původní tvar budov. Její původní hrana (a_1b_1) může být nahrazena buď hranami, které vzniknou prodloužením vnějších hran objektů (E_{a1}, E_{b1}) a jejich protnutím (E_1 na obr. 3.3b) nebo prodloužením a protnutím vnitřních hran I_{a1}, I_{b1} (I_1 na obr. 3.3b). Obdobně získáme dva nové body I_2 a E_2 pro nahrazení hrany a_2b_2. Kombinací těchto čtyř bodů můžeme získat čtyři různá řešení (E_1,E_2; E_1, I_2; I_1, E_2; I_1, I_2). Upřednostňovány jsou interní body, ale v případě porušení topologie nebo minimální šířky agregované oblasti může být vybrán externí bod tak, aby minimalizoval změnu plochy.

WARE et al., 1995 navrhli algoritmus pro agregaci pomocí Delaunay triangulace (obr. 3.4). Hrany vytvořené triangulace rozdělí prostor mezi agregovanými prvky, který je následně zaplněn.

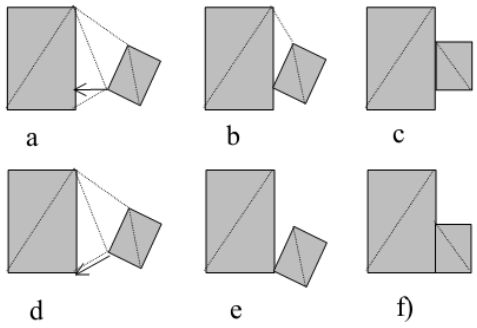
Zpráva dále uvádí dva algoritmy založené na posunu prvků. První z nich navrhli v roce 1995 WARE et al., jeho princip je znázorněn na obr. 3.5. Nejprve je zkonstruována Delaunay triangulace nad vrcholy agregovaných prvků. Dále je určena *úsečka posunu*, a to buď jako nejmenší vzdálenost mezi prvky (obr. 3.5a) nebo jako nejkratší hrana triangulace (obr. 3.5d). Podle hodnocení zprávy AGENT je z hlediska kartografické generalizace obecně vhodnější přisunutí objektů po úsečce určené nejkratší hranou triangulace.

Na základě účelové funkce (*cost function*) je spočteno, o kolik a jak se proporcionálně objekty posunou (buď oba, nebo jen jeden). Následně je provedeno vlastní přisunutí po úsečce tak, že se objekty buď střetnou hranou a vertexem (obr. 3.5b) nebo vertexem a vertexem (obr. 3.5e). Na závěr je provedena rotace objektů a jejich spojení (obr. 3.5c, 3.5f).



Obr. 3.4: Agregace přidáním plochy, WARE et al., 1995 (převzato z: AGENT, DD3, 2001, s. 39)

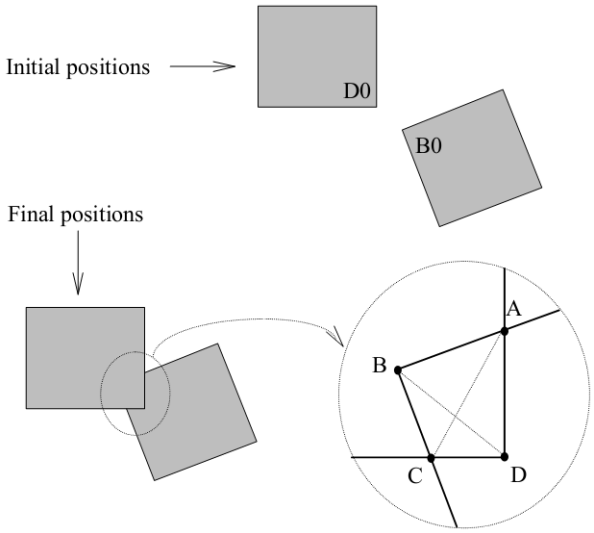
Také STAUFENBIEL, 1973 (podle ŠTAMPACH, 2006) navrhoval podobný přístup k agregaci budov. Po prostudování existujících map tvrdil, že přisunutí budov je nejčastějším případem agregace, pokud je úhel natočení menší než 45°. Sám však nenavrhl žádný konkrétní algoritmus, kterým této agregace dosáhnout.



Obr. 3.5: Agregace posunutím, WARE et al., 1995 (převzato z: AGENT, DD3, 2001, s. 38)

Druhý algoritmus založený na posunutí navrhl REGNAULD v roce 1998, jeho princip je na obr. 3.6. Agregace se dosáhne posouváním objektů, dokud se nepřekrývají. K posunu dochází po vektoru nejkratší vzdálenosti mezi objekty, která je určena buď jako nejkratší vzdálenost mezi vertexem a vertexem, vertexem a hranou nebo hranou a hranou. Míra posunu objektů je stejně jako u předchozího algoritmu určena účelovou funkcí. Překrývání objektů je zastaveno v okamžiku, kdy je splněna podmínka nejmenší šířky objektu (na obr. 3.6 spojnice AC).

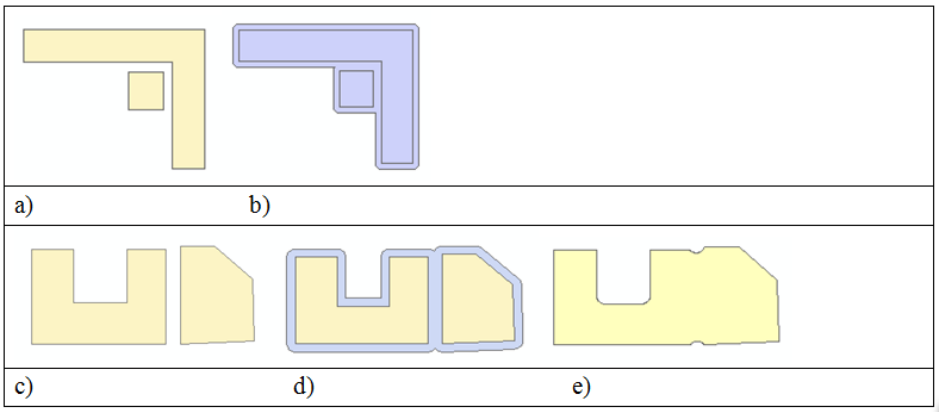
Oba tyto algoritmy výrazně modifikují umístění a strukturu původních objektů a ani nedávají příliš dobré kartografické výsledky. Dalším problémem může být sebeprotínání (*self-intersection*).



Obr. 3.6: Agregace posunutím, REGNAULD, 1998 (převzato z: AGENT, DD3, 2001, s. 38)

Problém agregace byl také řešen s využitím bufferu (např. ESRI, 2000). Tento přístup však není kartograficky příliš příznivý. Může být použit pouze jeden buffer, který by zaplnil mezery mezi agregovanými objekty, nicméně tím se neúměrně zvyšuje plocha agregovaných objektů. V manuálu ESRI je uveden příklad řešení jedním vnějším bufferem pro zaplnění mezer a následné použití vnitřního bufferu se stejným poloměrem aplikovaným na nově vzniklý objekt. Problémem je, že opakovaná aplikace bufferu poskytuje nepřirozené kartografické výsledky, viz obr. 3.7. Hlavními nevýhodami jsou:

- přílišné zvětšení plochy agregovaných oblastí,
- zaoblení pravých úhlů,
- vznik „obloučků“ v místě spojení původních polygonů.



Obr. 3.7: *(a, b)* agregace jednoduchým bufferem, *(c, d, e)* agregace vnějším a vnitřním bufferem (zpracování: ArcMap 9.3)

3.3.2 Region-based algoritmy

Algoritmus SU et al. využívá pro agregaci operátory matematické morfologie. Matematická morfologie vychází z teorie množin a je to věda o tvaru a struktuře geometrických struktur, vytvořená v 60. letech. Jejími autory jsou francouzští geostatistikové G. Matheron a J. Serra. Základní morfologické operátory dilatace a eroze, jsou definovány následovně:

$$\textit{Dilatace:} \qquad A \oplus B = \{a + b : a \in A, b \in B\} = \cup_{b \in B} A_b$$

$$\textit{Eroze:} \qquad A \Theta B = \{a : a + b \in A, b \in B\} = \cap_{b \in B} A_b$$

A je prvek, který chceme modifikovat a B je tzv. *strukturní element*, který se po A pohybuje konvolucí. V případě dilatace je oblast a rozšířena a v případě eroze jsou zachovány pouze ty pixely, nad kterými je strukturní element shodný s A.

Dalšími dvěma operátory jsou otevření a uzavření, které jsou složitější, jedná se o kombinaci dilatace a eroze. Jsou definovány takto:

$$\textit{Otevření:} \qquad A \circ B = (A \Theta B) \oplus B$$

$$\textit{Uzavření:} \qquad A \bullet B = (A \oplus B) \Theta B$$

SU et al. v návrhu svého algoritmu využili právě kombinace dilatace a eroze. Předkládají jednoduchý algebraický model:

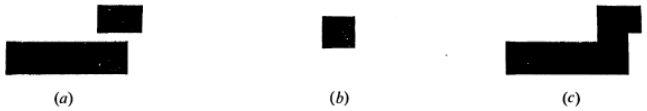
$$C = (A \oplus B_1) \Theta B_2$$

Pokud jsou oba strukturní elementy B_1 a B_2 shodné, jedná se o uzavření. Velikost a tvar strukturních elementů jsou určeny poměrem původního a nového měřítka (velikost) a obecným tvarem generalizovaných struktur (tvar – strukturní element je pak čtverec, úhlopříčka, kolečko atd). Ukázky funkčnosti předloženého algoritmu jsou na obr. 3.8 a 3.9.

Proces agregace v širším kontextu by měl být následován zjednodušením tvaru agregovaných polygonů (SU, 1997; LI, 2007). Po procesu agregace dojde ke spojení blízkých nebo sousedních polygonů, výsledný tvar však bude málokdy dostatečně zjednodušený. Budou se na něm vyskytovat malé konkávní nebo konvexní tvary, které v daném měřítku nejsou důležité. To ilustruje obr. 3.10.

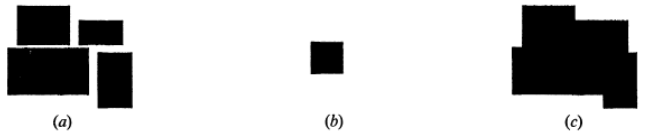
Funkcionalita algoritmu SU et al. z hlediska kartografické agregace na předvedených ukázkách je dobrá. Nicméně problémem tohoto algoritmu je komplikovaná změna typu datové reprezentace (vektor-rastr-vektor). Konverze vektor-rastr není tolik složitá, je však ztrátová. Převod z rastru na vektor je navíc daleko složitější a výpočetně náročnější. Problematickým se také může jevit výběr tvaru a velikosti strukturního elementu.

Ideální by tak bylo navrhnout algoritmus, který by pracoval pouze s vektorovými daty bez nutností vzájemných konverzí mezidatovým reprezentacemi.



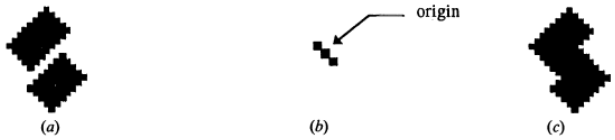
(a) (b) (c)

Figure 11. Combination of rectangular features (I). (*a*) Original features A, (*b*) Structuring element $B = B_1 = B_2$, (*c*) Combined area $C = A \cdot B$.



(a) (b) (c)

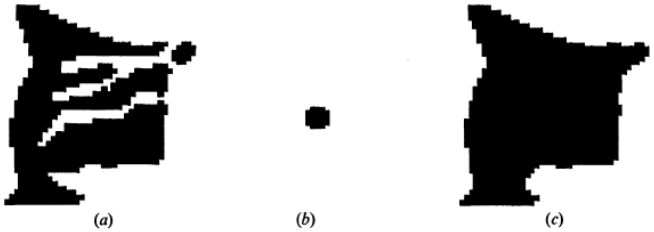
Figure 12. Combination of rectangular features (II). (*a*) Original features A, (*b*) Structuring element $B = B_1 = B_2$, (*c*) Combined area $C = A \cdot B$.



(a) (b) (c)

Figure 13. Combination of inclined rectangular features. (*a*) Original features A, (*b*) Structuring element $B = B_1 = B_2$, (*c*) Combined area $C = A \cdot B$.

Obr. 3.8: Příklad agregace pravoúhlých tvarů (převzato z: SU et al., 1997, s. 242)



(a) (b) (c)

Figure 16. Combination of curved feature area features (III). (*a*) Original features A, (*b*) Structuring element B, (*c*) Combined area $C = A \cdot B$.

(a) (b) (c)

Figure 17. Combination of curved area features (IV). (*a*) Original feature A, (*b*) Structuring element B, (*c*) Combined area $C = A \cdot B$.

Obr. 3.9: Příklad agregace složitějších tvarů (převzato z: SU et al., 1997, s. 243)

Obr. 3.10: Skupina polygonů vstupujících do algoritmu (*vlevo*), po agregaci (pro měřítko 10x menší, *uprostřed*) a následném zjednodušení tvaru (*vpravo*) (převzato z: SU et al., 1997, s. 241, 245)

4 POMOCNÉ DATOVÉ STRUKTURY A ALGORITMY

Navržený algoritmus bude používat kombinaci více technik a postupů. Uveďme zde stručně funkcionalitu algoritmu, aby bylo zřejmé, k jakému účelu budou níže uvedené struktury a algoritmy sloužit.

Algoritmus na úvod vyhledává budovy vhodné pro agregaci. V druhé fázi jsou budovy určené pro agregaci nahrazeny topologickou kostrou zkonstruovanou metodou straight skeletonu. Straight skeleton je následně redukován pouze na vnitřní hrany a vrcholy, nazvěme tuto strukturu jako redukovaný straight skeleton. Nalezením minimální kostry grafu, tvořeného dílčími redukovanými straight skeletony, je určen redukovaný skeleton polygonu reprezentujícího agregovanou budovu. Tuto strukturu je vhodné ještě generalizovat zjednodušením tvaru, např. odstraněním nevýznamných vrcholů. Posledním krokem je zpětná rekonstrukce agregovaného polygonu z generalizovaného redukovaného skeletonu.

Vlastní návrh algoritmu bude podrobně rozebrán v kapitole 6, zde budou z formálního hlediska popsány pomocné struktury a algoritmy.

4.1 Voronoi diagram

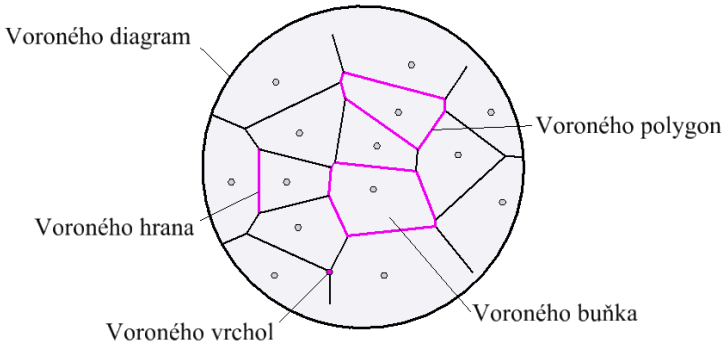
Voronoi diagram bude využit v první části algoritmu pro rychlé nalezení navzájem blízkých budov. Na základě sousednosti Voronoi buněk budou definovány sousedící polygony - ty budou podrobeny testování, jsou-li vhodné pro agregaci.

Definice. Označme euklidovskou vzdálenost mezi dvěma body p, q jako $dist(p, q)$. V rovině pak platí:

$$dist(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Nechť $P = \{p_1, p_2, \ldots, p_n\}$ je množina bodů v rovině, tyto body jsou nazývány *generátory*. Voronoi diagram $V(P)$ množiny P je definován jako rozdělení roviny do n buněk, jedna pro každý generátor p s takovou vlastností, že bod q leží v buňce odpovídající generátoru p_i právě tehdy, platí-li: $dist(q, p_i) < dist(q, p_j)$ pro $\forall p_j \in P, j \neq i$. (BERG et al., 2008, s. 148-149)

Voronoi diagram tedy rozděluje rovinu na Voronoi buňky, pro které platí, že každý bod v buňce má nejblíže k centru (tj. ke generátoru) této buňky, než k centru jiné buňky. Body ležící na hranách mají blíže k více než jednomu centru. Hrany jsou tvořeny úsečkami, polopřímkami nebo přímkami. Obecně mohou být generátory Voronoi buněk i vícerozměrné prvky (např. linie, kružnice).



Obr. 4.1: Voronoi diagram (upraveno podle: BAYER, 2008)

Konstrukce. V počítačovém prostředí jsou Voronoi diagramy konstruovány více metodami, mezi nejčastěji používané patří konstrukce Fortunovým algoritmem a inkrementálním vkládáním. Vytvořená teselace bývá ukládána v datové struktuře DCEL (*Doubly Connected Edge List*), která má navíc uložen ukazatel na generátor buňky.

Inkrementální vkládání. Algoritmus pracuje na principu postupného přidávání bodů do $V(P)$. Předpokládejme, že již máme vytvořený $V(P)$ pro prvních $j-1$ generátorů a chceme vložit generátor p_j (viz obr. 4.2). Jako první je nutné najít generátor p_i, v jehož buňce leží p_j. Pak je zkonstruován bisektor w_1w_2 mezi p_i a p_j kolmý na jejich spojnici tak, že p_j leží od bisektoru vlevo. Dále se pokračuje postupnou konstrukcí hran jako bisektorů p_j a každého z m generátorů jeho sousedních buněk. Pak tato sekvence hran $w_1w_2, w_2w_3, \ldots, w_mw_1$ tvoří hranici Voroného polygonu generátoru p_j. Na závěr se smažou vnitřní hrany v nově vniklé buňce.

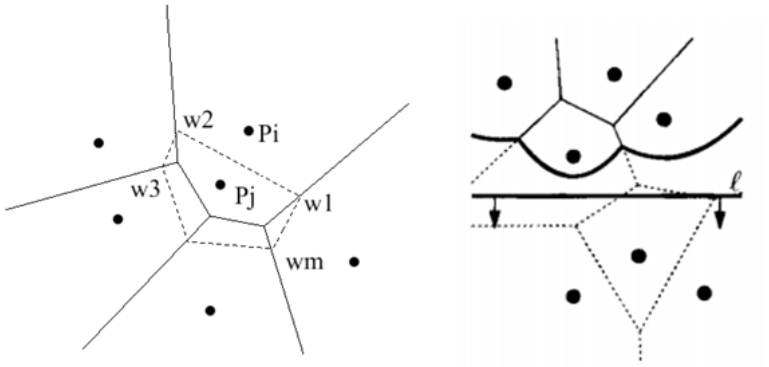
Problémem je vkládání generátorů, jejichž Voroného polygon má hrany končící v nekonečnu. Proto je na začátek celého algoritmu navíc vytvořen simplex tvořený třemi dalšími generátory p_1, p_2, p_3, které leží v dostatečné vzdálenosti od ostatních generátorů. Jejich souřadnice mohou být následující:

$$p_1 = [x_c, y_c + \frac{3\sqrt{2}}{2h}\ ]$$
$$p_2 = [x_c - \frac{3\sqrt{6}}{4h}\ , y_c - \frac{3\sqrt{2}}{4h}\ ]$$
$$p_3 = [x_c + \frac{3\sqrt{6}}{4h}\ , y_c + \frac{3\sqrt{2}}{4h}\ ]$$

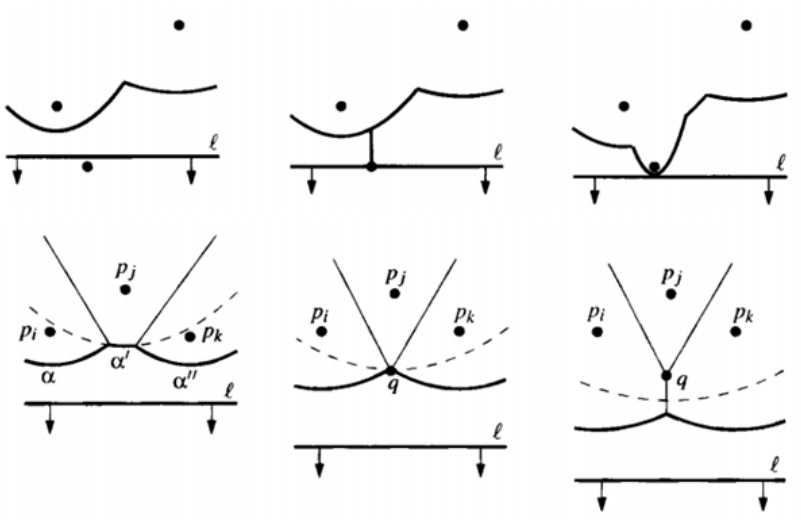
odvozené ze souřadnic min-max boxu

$$h = max(x_{max} - x_{min}, y_{max} - y_{min})$$

a těžiště $T[x_c, y_c]$ množiny P. (OKABE et al., 2000)



Obr. 4.2: Konstrukce VD inkrementálním vkládáním (*vlevo*), princip Fortunova algoritmu (*vpravo*) (převzato z: BERG et al., 2008, s. 152)



Obr. 4.3: site event (*nahoře*), circle event (*dole*) (převzato z: BERG et al., 2008, s. 153-154)

Fortunův algoritmus. Tento algoritmus bývá také označován jako *sweep line algoritmus* (sweep line = zametací přímka). Základním principem je posunování zametací přímky l rovinou odshora dolů a udržování informace o Voronoi diagramu nad přímkou l, který už nemůže být změněn generátory pod přímkou l. Označme polorovinu nad l jako l^+, polorovinu pod l jako l^-. Pro každý bod $q \in l^+$ a pro každý generátor $p_i \in l^-$ pak platí:

$$d(q, p_i) > d(q, l)$$

Nejbližší generátor p_i k bodu q pak leží v polorovině l^+, pokud platí:

$$d(q, p_i) \leq d(q, l)$$

Množina bodů q, které splňují tuto podmínku, tvoří parabolu. Pro více generátorů je tato množina tvořena parabolickými oblouky nazývanými *beach line*, viz obr. 4.2. Zlomové body beach line pak kopírují hrany Voronoi diagramu. Vertexy a nové hrany Voronoi diagramu vznikají pohybem zametací přímky, který způsobuje změnu topologie beach line. Může dojít ke vzniku dvou událostí:

- ***site event*** - zametací přímka dorazí k dalšímu generátoru, čímž se vytváří nový parabolický oblouk definující hranu Voronoi buňky (obr. 4.3 nahoře)
- ***circle event*** - dochází k zániku parabolického oblouku, čímž vzniká bod q jako nový vertex Voronoi diagramu (obr. 4.3 dole). Pro tento bod platí

$$d(q, p_i) = d(q, p_j) = d(q, p_k) = d(q, l)$$

  je tedy středem kružnice tvořené body p_i, p_j, p_k. (BERG et al., 2008)

Časová složitost. Algoritmus inkrementálním vkládáním má horní časovou složitost $O(n^2)$, Fortunův algoritmus $O(n \log n)$.

Struktura DCEL. Struktura DCEL slouží k uložení geometrických, topologických a případných dalších informací o rozdělení roviny. Obsahuje tři kolekce záznamů: jednu pro vrcholy, druhou pro hrany a třetí pro plochy. Každá hrana je vpodstatě uložena dvakrát ve formě tzv. *half-edges* (polohran). Jedná se o navzájem opačně orientované úsečky $\vec{e}$ a $Twin(\vec{e})$. Můžeme tak mluvit o jejich počátečním a koncovém vrcholu, kdy $Origin(\vec{e}) = Destination(Twin(\vec{e}))$ a naopak. Jejich orientace je taková, že vnitřek plochy (*face*), kterou ohraničují, leží vždy vlevo od polohrany. Díry mají orientaci opačnou. Geometrické a topologické vlastnosti rozdělení roviny jsou pak v DCEL uloženy následovně:

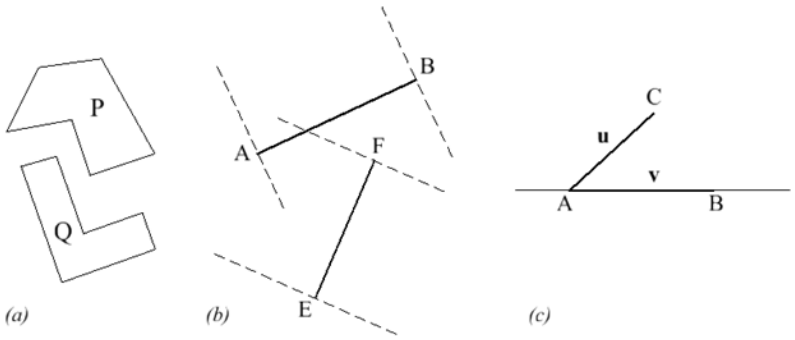
- záznam vrcholu v nese informace o svých souřadnicích a pointer na polohranu, jíž je počátečním bodem,
- záznam plochy f nese pointer na nějakou svoji vnější polohranu, v případě neuzavřené plochy je hodnota pointeru NULL. Dále má uložen seznam vnitřních komponent (děr), pro každou díru je udržován pointer na nějakou její polohranu,
- záznam polohrany $\vec{e}$ má uložen ukazatel k počátečním bodu polohrany $Origin(\vec{e})$, ukazatel na opačnou polohranu $Twin(\vec{e})$ a pointer na incidující plochu. Dále jsou uloženy ukazatele na předchozí a následující hrany ležící na hranici incidující plochy. (BERG et al., 2008)

Využití Voronoi diagramů. V kartografii a GIS jsou Voronoi diagramy (stejně jako Delaunay triangulace, s nimiž mají těsný vztah) často využívány jako pomocné geometrické struktury. V rámci GIS se jedná především o určování spádových oblastí, nalezení všech sousedů dané oblasti nebo o interpolaci metodou natural neighbour. V rámci kartografické generalizace jsou voronoi diagramy jedním z možných způsobů určení geografického kontextu (např. Basaraner a Selcuk, 2004) nebo určení sousednosti (např. Steihauer et al., 2001, podle LI et al., 2004).

4.2 Minimální vzdálenost dvou polygonů

Po určení dvou budov jako navzájem sousedících bude algoritmus testovat, má-li dojít k jejich agregaci. Jedním z hlavních kritérií je minimální vzdálenost mezi nimi. Otázka výpočtu vzdálenosti je řešena využitím analytické geometrie.

Označme oba analyzované polygony P, Q, viz obr. 4.4a. Problém vzájemné polohy je převeden na výpočet minimální vzdálenosti dvou úseček AB, EF, kde $AB \in P$ a $EF \in Q$, které tvoří hrany testovaných polygonů (obr. 4.4b).



Obr. 4.4: Výpočet vzdálenosti dvou polygonů

Určení orientace vektorů. Označme vektor $u = \overrightarrow{AC} = (u_x, u_y)$, vektor $v = \overrightarrow{AB} = (v_x, v_y)$. Pokud je úhel mezi vektory $\vec{u}$, $\vec{v}$, měřený proti směru hodinových ručiček, menší než π, pak je orientace vektorů kladná, pokud je větší než π, je orientace vektorů záporná. Výpočet orientace dvou vektorů lze využít pro analýzu vzájemné polohy bodu a přímky.

Nechť C představuje bod, jehož polohu analyzujeme vzhledem k přímce určené body AB, viz obr. 4.4c. Pak platí

$$t = \begin{vmatrix} u_x & u_y \\ v_x & v_y \end{vmatrix} \quad \begin{cases} t < 0 & C \text{ leží vlevo od } AB \\ t = 0 & C \text{ leží na } AB \\ t > 0 & C \text{ leží vpravo od } AB \end{cases}$$

Určení vzdálenosti dvou úseček. Minimální vzdálenost dvou úseček AB, EF (viz obr. 4.4b) lze určit z následujícího vztahu:

$$d_{min}(AB, EF) = min(d(A, EF), d(B, EF), d(E, AB), d(F, AB))$$

Při výpočtu vzdálenosti úseček je třeba rozlišit, zda testovaný bod (vezměme např. případ $d(A, EF)$ leží v pásu vymezeném normálami úsečky EF, nebo v některém z obou krajních pásů. K tomu je využit test orientace vektorů $\overrightarrow{n_{EF}} = (-u_y, u_x)$, $\overrightarrow{EA}$ a $\overrightarrow{n_{EF}}$, $\overrightarrow{FA}$. Pokud

$$\begin{aligned}
&t(\vec{n}, \overrightarrow{EA}) < 0 \text{ AND } t(\vec{n}, \overrightarrow{FA}) < 0, \quad A \in \varepsilon_L \text{ (levý pás)} \\
&t(\vec{n}, \overrightarrow{EA}) > 0 \text{ AND } t(\vec{n}, \overrightarrow{FA}) < 0, \quad A \in \varepsilon_S \text{ (střední pás)} \\
&t(\vec{n}, \overrightarrow{EA}) > 0 \text{ AND } t(\vec{n}, \overrightarrow{FA}) > 0, \quad A \in \varepsilon_P \text{ (pravý pás)}
\end{aligned}$$

Vzdálenost $d(A, EF)$, kde $A = [x_A, y_A]$ je pak vypočtena následovně. Pokud

$$\begin{aligned}
A \in \varepsilon_L &: d(A, EF) = |\overrightarrow{EA}| \\
A \in \varepsilon_S &: d(A, EF) = \frac{|ax_a \cdot by_a \cdot c|}{\sqrt{a^2 + b^2}} \\
A \in \varepsilon_P &: d(A, EF) = |\overrightarrow{FA}|
\end{aligned}$$

kde a, b, c jsou koeficienty obecné rovnice přímky určené body EF.

Určení vzdálenosti dvou polygonů. Minimální vzdálenost mezi polygony P, Q je pak určena následovně:

$$d_{min}(P, Q) = min \begin{pmatrix} min(d_{min}(p_0p_1, q_0q_1), \ldots, d_{min}(p_{i-1}p_0, q_0q_1)), \\ \ldots \\ min(d_{min}(p_0p_1, q_{j-1}q_0), \ldots, d_{min}(p_{i-1}p_0, q_{j-1}q_0)) \end{pmatrix}$$

kde i je počet vrcholů polygonu P, j je počet vrcholů polygonu Q.

Časová složitost. Časová složitost algoritmu je $O(m \cdot n)$, kde m je počet vrcholů polygonu P, n je počet vrcholů polygonu Q.

4.3 Straight skeleton

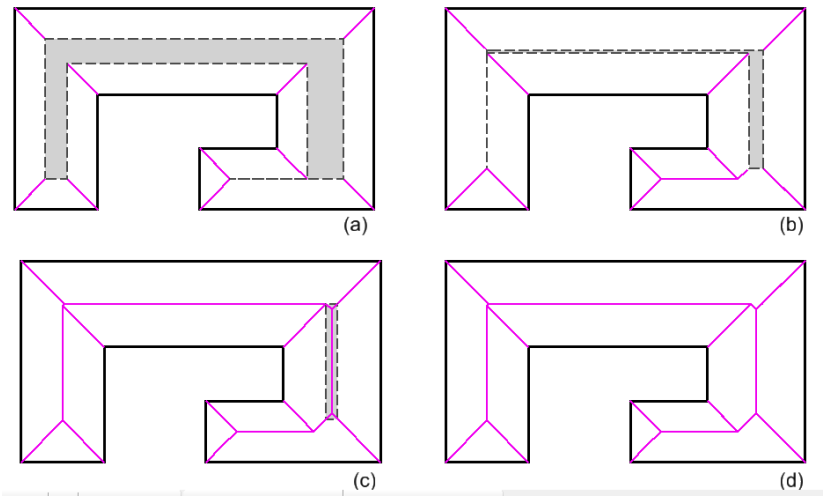
Jak již bylo uvedeno výše, straight skeleton bude využit nejprve pro redukci polygonů budov na jejich topologickou kostru. V této souvislosti definujeme novou datovou strukturu zvanou redukovaný straight skeleton (viz část 4.4).

Závěrečným krokem navrženého algoritmu je zpětná rekonstrukce polygonu z redukovaného straight skeletonu. Jedná se o inverzní postup ke konstrukci straight skeletonu, který bude využívat datové struktury a funkcí algoritmu Felkela a Obdržálka, 1998.

Topologická kostra. Definujme uzavřený ohraničený polygon P v R^2. Topologická kostra je výsledkem procesu zvaného *skeletonizace*. Skeletonizace provádí dekompozici P na posloupnost jednodušších 1D entit.

Straight skeleton, představený O. AICHHOLZEREM a F. AURENHAMMEREM, 1996, je jednou z metod konstrukce topologické kostry polygonu. Topologická kostra je prostorovou redukcí prvku tvořenou posloupností úseček či křivek, u straight skeletonu pouze úseček. Zachovává tvarové a geometrické charakteristiky polygonů, a je tak vhodnou strukturou pro automatizovanou kartografickou generalizaci (BAYER, 2008).

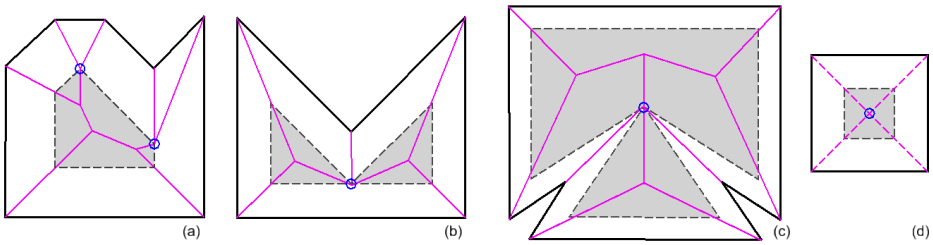
Definice. Definujme tzv. *smršťovací proces*. Jedná se o proces, kdy se posunují vrcholy polygonu P po osách úhlů (*bisektorech*) incidujících hran směrem dovnitř konstantní rychlostí, dokud není plocha polygonu nulová. Straight skeleton $S(P)$ je pak definován jako sjednocení částí os úhlů (tj. bisektorů), po kterých se pohybují vrcholy P během smršťovacího procesu. Tyto úsečky jsou označovány jako *segmenty* (*hrany*) skeletonu, jejich koncové body, které nejsou vrcholy P, jsou *uzly* skeletonu. Každý uzel odpovídá události *edge*, *split* nebo *vertex event*. Oblasti vymezené hranicí polygonu a segmenty skeletonu jsou nazývány *ploškami* (*faces*).



Obr. 4.5: Princip tvorby straight skeletonu

Konstrukce. Při smršťovacím procesu dochází k různým změnám v topologii polygonu. Tyto změny mohou být trojího typu, jejich ilustrace je na obr. 4.6:

- ***zánik hrany*** (***edge event***) - délka hrany se smršťováním zmenší na nulu (tj. bod) a způsobí, že její incidující hrany se stanou navzájem sousedícími,
- ***rozdělení hrany*** (***split event***) - vnitřní nekonvexní úhel se střetne s posunující se protilehlou hranu polygonu a dojde tak k jejímu rozdělení,
- ***vertex event*** - událost definovaná až v roce 1999 EPPSTEINEM a ERICKSONEM, při níž dochází ke střetnutí dvou nebo více nekonvexních vrcholů polygonu.



Obr. 4.6: *(a)* dva současné edge eventy, *(b)* split event, *(c)* vertex event, *(d)* degenerovaný straight skeleton (upraveno podle: EPPSTEIN a ERICKSON, 1999, s. 8)

V případě *edge event* nebo *split event* vznikne jedna nebo více nových konvexních podoblastí. V případě vertex event dojde k rozpadu na dvě nebo více podoblastí, z nichž některé mohou být opět nekonvexní. Podoblasti jsou zdrojem nových os úhlů a jsou dále zpracovávány. V případě pravidelných n-úhelníků je skeleton degradován na jediný uzel vysšího stupně (obr. 4.6d).

Pořadí vzniku událostí. Klíčovým problémem správné konstrukce straight skeletonu je určení pořadí vzniku jednotlivých událostí. Vznik událostí (a tedy i uzlů skeletonu) je jednoznačně určen délkou pohybu vrcholu po ose úhlu. Události jsou ukládány v prioritní frontě, která zajišťuje správné pořadí jejich zpracování. Algoritmy, které řeší konstrukci straight skeletonu, se pak liší především způsobem detekce a udržováním událostí.

Implementace algoritmu. Algoritmus pro konstrukci straight skeletonu implementovali např. AICHHOLZER a AURENHAMMER, 1996; FELKEL a OBDRŽÁLEK, 1998 nebo CACCIOLA, 2004. Aichholzer a Aurenhammer využívají triangulaci a prioritní frontu. Pro vnitřek zmenšovaného polygonu je provedena triangulace. Zánik trojúhelníka vede ke změně topologie. Časy zániku trojúhelníků jsou uloženy v prioritní frontě, která je postupně zpracovávána. Felkel a Obdržálek využívají datovou strukturu SLAV (*set of circular lists of active vertices*), která uchovává *ukazatele* (pointery) na hrany původního polygonu. Cacciola implementoval straight skeleton pro open source knihovnu CGAL (*Computional Geometry Algorithms Library*). Vychází z algoritmu Felkela a Obdržálka, je však obohacena o vertex event, který v původní implementaci chybí. Využívá datové struktury HDS (*Halfedge Data Structure*), která je součástí CGAL knihovny.

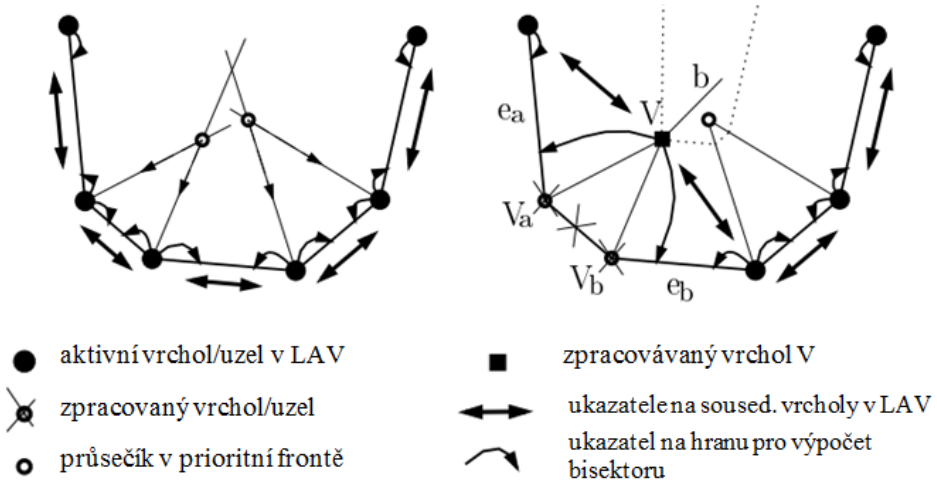
Algoritmus Felkela a Obdržálka

Jak již bylo uvedeno, algoritmus využívá datové struktury SLAV (*set of circular list of active vertices - sada kruhových seznamů aktivních vrcholů*). Tato struktura je tvořená kruhovými seznamy vrcholů (LAV) pro každý polygon, díru polygonu a polygony vzniklé v průběhu skeletonizace. Konvexní polygon je tak představován pouze jednou strukturou LAV, nekonvexní polygon nebo polygon s dírami více strukturami. Každý vrchol ve SLAV má přiřazeny ukazatele na oba sousedící (tj. předchozí a následující) vrcholy v seznamu (LAV). Konstrukce se liší pro konvexní a nekonvexní polygony. V případě konvexních polygonů může dojít pouze k *edge event*, v případě nekonvexních i ke *split event* (jak bylo uvedeno, algoritmus nezahrnuje *vertex event*).

Konstrukce pro konvexní polygony

Za předpokladu, že vrcholy a hrany polygonu jsou orientovány proti směru hodinových ručiček, probíhá algoritmus následovně.

1. *Inicializace:*
   (a) vlož všechny vrcholy $V_1, V_2, \ldots, V_n$ do dvojitého kruhového seznamu (LAV) uloženého ve SLAV. Všechny vrcholy jsou označeny jako aktivní - nezpracované.
   (b) pro každý vrchol V_i v LAV přidej pointery (ukazatele) na jeho incidentní hrany $e_{i-1} = V_{i-1}V_i$ a $e_i = V_iV_{i+1}$, a vypočti osu úhlu b_i těmito hranami sevřeného.
   (c) pro každý vrchol V_i vypočti nejbližší průsečík jeho osy úhlu b_i s osami úhlů b_{i-1}, b_{i+1} přilehlých vrcholů a pokud existuje, ulož ho do prioritní fronty v závislosti na vzdálenosti od hrany e_i. Pro každý průsečík I_i ulož ukazatele na vrcholy V_a, V_b, z nichž vychází protínající se osy úhlů.
2. *Dokud není prioritní fronta s průsečíky I prázdná, prováděj následující:*
   (a) vezmi průsečík I s největší prioritou,
   (b) pokud jsou vrcholy/uzly V_a a V_b, na které ukazuje I, označené jako zpracované, pokračuj krokem 2, pokud ne, dochází k *edge event* a hrana zaniká:
   (c) pokud v LAV platí, že předchůdce vrcholu předcházející V_a odpovídá vrcholu V_b, přidej do skeletonu tři segmenty V_aI, V_bI, V_cI, kde V_c je předchůdce V_a a zároveň následník V_b (v LAV zbývají poslední tři vrcholy) a pokračuj krokem 2. V opačném případě přidej pouze hrany V_aI, V_bI a modifikuj LAV následovně:
      - označ vrcholy V_a, V_b jako zpracované,
      - vytvoř nový vrchol V na místě I, vlož ho do LAV a přidej ukazatele na přilehlé hrany e_a, e_b,
      - spočti osu úhlu b mezi e_a, e_b a vypočti a ulož průsečík I stejně jako v kroku 1c).

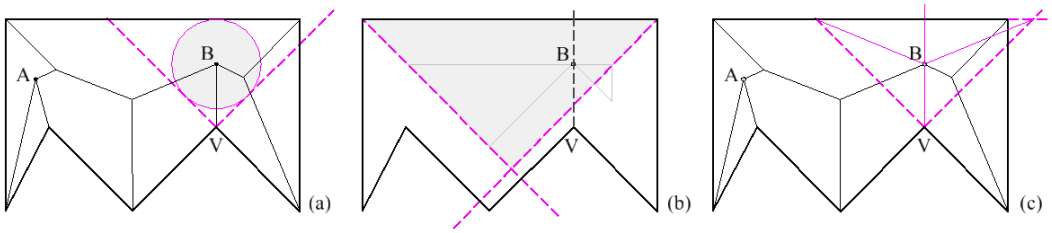


Obr. 4.7: Datová struktura LAV (upraveno podle: FELKEL a OBDRŽÁLEK, 1998, s. 4)

Konstrukce pro nekonvexní polygony

Princip konstrukce je podobný konstrukci pro konvexní polygony. Pro průsečíky I v prioritní frontě je navíc ukládána informace o tom, zda se jedná o *split event* nebo *edge event*. Přítomnost nekonvexního vrcholu může vést k rozdělení hrany, ale nemusí. Na obr. 4.8a bod A představuje *edge event*, kdežto bod B *split event*. Hlavním problémem je určení souřadnic bodu B. Ten má tu vlastnost, že leží ve stejné vzdálenosti od protilehlé hrany vrcholu V i od prodloužení hran, které incidují s vrcholem V (obr. 4.8a). Problém je, že neznáme protilehlou hranu e. Pro hrany se testuje existence průsečíku hrany e_i, resp. přímky, která hranu obsahuje, s bisektorem úhlu vrcholu V a zda potenciální bod B_i leží v oblasti vymezené hranou e_i a bisektory vycházejících z incidujících vrcholů (obr. 4.8b). Samotný vrchol B_i je určen jako průsečík os úhlů trojúhelníka, který vznikne z hrany e_i (resp. její přímky) a prodloužení hran incidujících s vrcholem V (obr. 4.8c). Výsledný bod B je pak vybrán z množiny bodů B_i jako ten s nejmenší vzdáleností od V.

Pokud je určeno, že bod B odpovídá události *split event*, je nutné provést změnu v LAV, neboť dojde k rozdělení polygonu na dvě části. LAV se rozpadne na dvě nové struktury a do každé jsou souřadnice bodu B uloženy jako nové uzly.



Obr. 4.8: Konstrukce straight skeletonu pro nekonvexní polygony (upraveno podle: FELKEL a OBDRŽÁLEK, 1998, s. 5)

1. *Inicializace:*

   (a) proveď stejné operace jako v případě konvexních polygonů. Pro nekonvexní vrcholy navíc spočti průsečíky s protilehlými hranami a do prioritní fronty ulož průsečík I s informací o typu události.

2. *Dokud není prioritní fronta prázdná, prováděj následující:*

   (a) vezmi průsečík I z prioritní fronty. Pokud se jedná o *edge event*, pokračuj jako v případě konvexních polygonů. V opačném případě postupuj v tomto algoritmu.

   (b) pokud je vrchol/uzel V, na který ukazuje I, označený jako zpracovaný, pokračuj krokem 2, v opačném případě přidej ke kostře hranu VI a proveď změny ve SLAV:

   - označ vrchol V jako zpracovaný,
   - LAV se rozpadne na dvě části, do každé vlož jeden vrchol (V_1, V_2) se souřadnicemi shodnými s bodem I. Vrchol V_1 vlož mezi předchůdce V a koncový bod protilehlé hrany, vrchol V_2 vlož mezi následníka V a počáteční bod protilehlé hrany, čímž dojde k rozdělení polygonu na dvě části. Pro oba vrcholy ulož ukazatele na odpovídající hrany,
   - pro vrcholy V_1, V_2 spočítej bisektory a odpovídající průsečíky a ulož je do prioritní fronty jako v kroku 1c) společně s typem události.

Časová složitost. Časová složitost algoritmu je $O(n \cdot m + n \log n)$, kde n je celkový počet vrcholů polygonu a m je počet nekonvexních vrcholů.

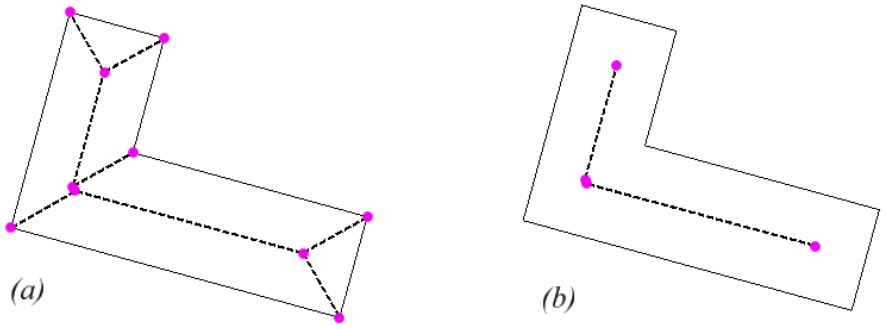
Využití straight skeletonu. V oblasti kartografické generalizace lze skeleton využít např. při generalizaci prostorovou redukcí (např. generalizace říčních toků nebo cestních sítí, HAUNERT a SESTER, 2008) nebo rozdělení polygonu polygonům sousedním (ŠÍP, 2007). Mezi další aplikace patří např. nástroje pro umisťování popisů nebo rekonstrukci tvarů střech.

4.4 Redukovaný straight skeleton

Redukovaný straight skeleton $RS(P)$ je nově definovanou strukturou, která je podmnožinou straight skeletonu $S(P)$. V algoritmu bude využit jako vstup pro nalezení minimální kostry agregovaného polygonu. Nalezená minimální kostra je pak redukovaným straight skeletonem agregovaného polygonu. Z redukovaného skeletonu bude provedena zpětná rekonstrukce agregovaného polygonu.

Definice. Nechť $S(P) = v, e$ je straight skeleton polygonu P, kde v je množina vrcholů skeletonu, e je množina hran skeletonu. Označme v_I podmnožinu množiny v takovou, že je tvořena pouze vnitřními vrcholy skeletonu (uzly). Obdobně označme e_I podmnožinu množiny e takovou, že je tvořena pouze vnitřními hranami skeletonu. Redukovaný straight skeleton $RS(P)$ polygonu P pak definujme jako:

$$RS(P) = \{v_I, e_I\}$$



Obr. 4.9: *(a)* straight skeleton, *(b)* redukovaný straight skeleton

Konstrukce. Základem konstrukce redukovaného straight skeletonu je konstrukce straight skeletonu (viz kap. 4.3). Přidáním pomocného booleovského atributu (např. *inner*) uzlům konstruovaného straight skeletonu můžeme odlišit jeho vnitřní a vnější uzly:

$$\begin{aligned} \text{pro vnitřní uzly} \quad & inner = true \\ \text{pro vnější uzly} \quad & inner = false \end{aligned}$$

$\forall$ uzly u s atributem $inner == true$ a $\forall$ hrany e spojující uzly u tvoří hledaný redukovaný straight skeleton.

4.5 Grafové algoritmy

Grafové algoritmy, zejména algoritmy pro nalezení minimální kostry grafu a prohledávání do hloubky, budou v navrženém algoritmu jedněmi ze stěžejních technik. Minimální kostra grafu bude využita pro konstrukci redukovaného skeletonu agregovaného polygonu z jednotlivých redukovaných skeletonů agregovaných budov. Prohledávání do hloubky se uplatní při větším počtu dílčích operací. Mezi ně bude patřit především určení skupin polygonů pro agregaci a určení správného řazení a orientace vrcholů redukovaného skeletonu agregovaného polygonu.

4.5.1 Minimální kostra grafu

Z hlediska teorie grafů můžeme na skeleton hledět jako na neorientovaný graf, navíc strom. S grafy je svázáno velké množství pojmů, zde uveďme ty, které mají přímou souvislost s řešením diplomové práce: neorientovaný graf, spojová reprezentace grafu, diskrétní graf, podgraf, faktor grafu, souvislost, les, strom, kostra grafu, minimální kostra grafu. Některé ze struktur ilustruje obr. 4.10.

Neorientovaný graf. Nechť E, V jsou libovolné disjunktní množiny a $\varepsilon : E \mapsto V \otimes V$ zobrazení. $V \otimes V$ je množina všech neuspořádaných dvojic $[u, v]$; u, $v \in V$. Neorientovaným grafem nazýváme uspořádanou trojici $G = (V, E, \varepsilon)$, prvky množiny E jsou hranami grafu G, prvky množiny V uzly grafu G a zobrazení ε incidencí grafu G. Incidence ε grafu přiřazuje každé jeho hraně neuspořádanou dvojici uzlů. (KOLÁŘ, 2004, s. 18)

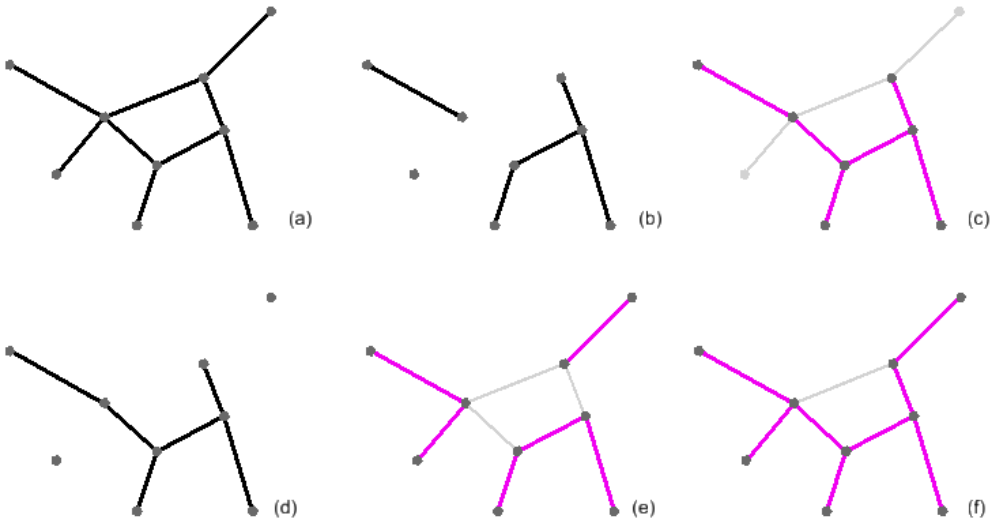
Spojová reprezentace grafu. Základní forma spojové reprezentace neorientovaného grafu $G = (V, E, \varepsilon)$ sestává z pole Adj obsahující $|V|$ odkazů na seznamy sousedů jednotlivých uzlů z množiny V. Pro každý uzel $v \in V$ obsahuje spojový seznam $Adj[v]$ jeden záznam pro každou neorientovanou hranu $e = [v, w] \in E$ s druhým krajním uzlem w. Záznam typicky obsahuje identifikaci (číslo) uzlu w. (KOLÁŘ, 2004, s. 72)

Diskrétní graf. Diskrétní graf je takový graf, jehož množina E je prázdná.

Podgraf. Graf G' je podgrafem grafu G, vznikne-li z grafu G vynecháním nějakých (nebo žádných) vrcholů a hran. Podgraf musí být také grafem: spolu s každou hranou, která je v podgrafu, tam musí být i její krajní vrcholy. (DEMEL, 2002, s. 18)

Faktor grafu. Faktor grafu G' je speciálním podgrafem grafu G, který vznikne vynecháním některých (nebo žádných) jeho hran, tj. platí-li $V(G) = V(G')$. (DEMEL, 2002, s. 18)

Souvislost, komponenta souvislosti. Graf nazýváme souvislým, jestliže každé jeho dva vrcholy jsou spojeny neorientovanou cestou. Komponenta souvislosti grafu G je každý podgraf H grafu G, který je souvislý a který je maximální s touto vlastností, tj. není částí většího souvislého podgrafu. (DEMEL, 2002, s. 57)



Obr. 4.10: *(a)* Neorientovaný graf, *(b)* podgraf, *(c)* cesta grafu, *(d)* faktor grafu, *(e)* les tvořený stromy (komponentami souvislosti), *(f)* kostra grafu

Les a strom. *Les* je graf, který neobsahuje kružnici. *Kružnice* je neorientovaná uzavřená cesta. *Cesta* je neorientovaný sled, v němž se neopakuje žádný vrchol. *Sled* je posloupnost vrcholů a hran $v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$, kde každá hrana e_i spojuje vrcholy v_{i-1}, v_i. *Strom* je graf, který neobsahuje kružnici a navíc je souvislý. (DEMEL, 2002, s. 19, s. 20, s. 58)

Kostra grafu. Faktor grafu G, který je stromem, nazýváme *kostrou grafu*. Každý souvislý graf má kostru. (DEMEL, 2002, s. 58)

Minimální kostra grafu. Nechť je dán souvislý neorientovaný graf $G = \langle E, V \rangle$ s nezáporným ohodnocením hran $\omega : E \mapsto R^+$. Problém minimální kostry grafu G spočívá v nalezení takové jeho kostry $T = \langle E_u, V \rangle$, která splňuje následující podmínku: $\sum_{e \in E_u} \omega(e)$ je minimální. (KOLÁŘ, 2004, s. 98)

Hledat minimální kostru grafu má smysl pouze tehdy, jsou-li hrany grafu ohodnocené. Všechny kostry daného grafu totiž obsahují stejný počet hran:

$$n_H = |u| - 1$$

kde u je počet vrcholů. Pokud nejsou hrany ohodnocené, nelze určit, která z nich je nejvhodnější pro daný účel (např. minimální). Nicméně i v případě ohodnocených hran může dojít k tomu, že minimální kostra není jednoznačně určena.

Hledání minimální kostry grafu spojováním podkoster s minimálním ohodnocením je blízké postupu agregace. Navržený algoritmus právě krokem hledání minimální kostry grafu agregaci provádí (viz část 6.2).

Využití. Úlohy, které využívají hledání minimální kostry, mají za cíl spojit množství bodů (míst) cestou nejmenších nákladů. Příkladem může být oblast dopravy (např. nalezení nejkratší cesty pro posyp silnic ve městě), energetiky (např. návrh sítě pro rozvod elektřiny - tuto úlohu řešil O. Borůvka při návrhu elektrických sítí v polovině 20. let 20. století) nebo telekomunikací.

Hledání minimální kostry grafu. Z algoritmického hlediska lze ke konstrukci minimální kostry (*Minimum Spanning tree - MST*) přistupovat více způsoby. Všechny varianty pracují iteračně: postupně konstruují podgrafy daného grafu tak, aby se přibližovaly hledané minimální kostře. Přibližování se může dosáhnout buď vypouštěním hran výchozího grafu, přidáváním hran k podgrafu, který je na počátku prázdný, nebo výměnou hran v nějaké výchozí kostře.

Z hlediska efektivity je nejvýhodnější varianta přidávání hran k podgrafu. Začíná se prázdnou množinou hran T, ke které se přidávají další vhodné hrany. Tento přístup ilustruje alg. 4.1. Základní podmínkou je, že množina hran T je podmnožinou hran nějaké minimální kostry - tato podmínka se nesmí přidáním nové hrany porušit. (KOLÁŘ, 2004, s. 100)

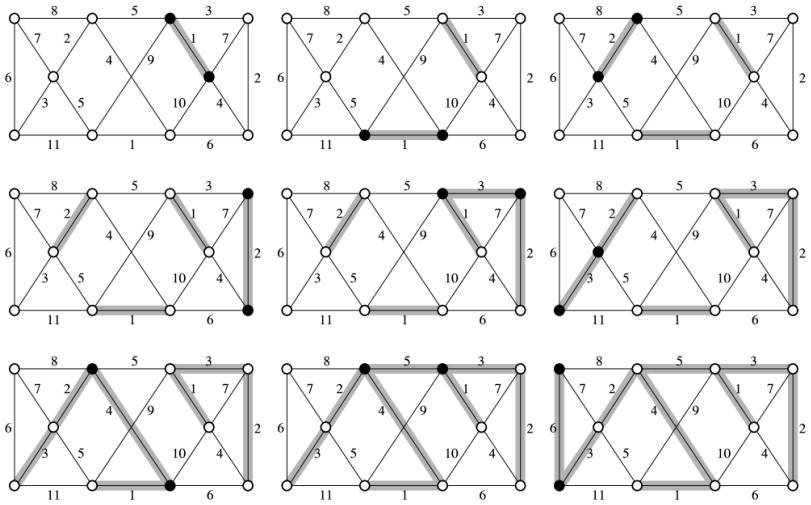
Algoritmus 4.1 - hledání minimální kostry grafu

1. $T = \emptyset$
2. `while` T `netvoří kostru`
3. `do najdi vhodnou hranu` $[u, v]$ `pro` T
4. $T = T \cup \{[u, v]\}$
5. `return` T

Základem úspěšného nalezení minimální kostry je grafu je zvolení hrany $[u, v]$, která je přidávána ke vznikající kostře (alg. 4.1, krok 3). Tento krok je řešen dvěma způsoby. Jako první navrhl algoritmus pro hledání minimální kostry již v roce 1926 O. Borůvka, v roce 1956 ho v souvislosti s rozvojem výpočetní techniky znovuobjevil američan J. Kruskal. Dnes je tento algoritmus označován za Borůvkův-Kruskalův. Efektivnější metodu řešení navrhl v roce 1930 V. Jarník a nezávisle na něm v roce 1957 R. C. Prim. Tento algoritmus tak bývá označován jako Jarníkův-Primův. Při řešení diplomové práce byl využit Kruskalův algoritmus, na tomto místě proto uvedeme pouze jeho princip.

Borůvkův-Kruskalův algoritmus

Princip algoritmu. Princip ilustruje obr. 4.11 a alg. 4.2. Základní myšlenkou je přidání takové hrany $E = [u, v]$, která má ze všech hran spojujících dva podstromy vznikající kostry minimální ohodnocení ω. U vybrané hrany je zjišťováno umístění incidujících uzlů. Na základě umístění uzlů je pak provedeno spojení odpovídajících podstromů.



Obr. 4.11: Princip Borůvkova-Kruskalova algoritmu (převzato z: KOLÁŘ, 2004, s. 102)

Algoritmus 4.2 - Borůvkův-Kruskalův algoritmus

1. $T = \emptyset$
2. `for` $\forall u \in U$
3. `do MAKE-SET(`u`)`
4. `uspořádej` E `do neklesající posloupnosti podle váhy` ω
5. `for` $\forall e = [u, v] \in E$ `v pořadí neklesajících vah`
6. `do if FIND-SET(`u`)` $\neq$ `FIND-SET(`v`)`
7. `then` $T = T \cup \{[u, v]\}$
8. `UNION(`u, v`)`
9. `return` T

`MAKE-SET(x)` vytvoří nový podgraf, jehož jediným prvkem (a zároveň i reprezentantem) je prvek x. Předpokládá se přitom, že x není prvkem žádného jiného podgrafu.

1. $p[x] = x$ `//nastavení otce ve stromu`
2. $hod[x] = 0$ `//udává vzdálenost k nejvzdálenějšímu listu stromu`

`UNION(x,y)` provede sjednocení podgrafů obsahujících prvky x a y. Tyto podgrafy jsou před sjednocením disjunktní. Důsledkem je i to, že se původní podgrafy odstraní a místo nich se vytvoří jeden sjednocený podgraf. Reprezentantem sjednocení může být jakýkoliv prvek.

1. `LINK(FIND-SET(x), FIND-SET(y))`

`LINK (x,y)`

1. `if` $hod[x] > hod[y]$ `then` $p[y] = x$
2. `else` $p[x] = y$
3. `if` $hod[x] = hod[y]$ `then` $hod[y] = hod[y] + 1$

`FIND-SET(x)` vrací jako svou hodnotu reprezentanta podgrafu obsahující prvek x.

1. `if` $x \neq p[x]$ `then` $p[x] =$ `FIND-SET(`$p[x]$`)`
2. `return` $p[x]$

Časová složitost. Časovou složitost lze stanovit rovnou $O(n \log n)$, kde n je počet hran grafu. (KOLÁŘ, 2004)

4.5.2 Prohledávání grafu do hloubky (Depth First Search)

Tento algoritmus je využíván pro analýzu grafu, jeho cílem je dostat se do grafu co „nejhlouběji“. Algoritmus prochází hrany vycházející z posledně nalezeného uzlu u, který ještě má nějaké neprozkoumané hrany. Když projde všechny jeho hrany, vrátí se zpátky k uzlu, ze kterého se do uzlu u dostal a z něho pokračuje po další dosud neprozkoumané hraně. Postupuje se tak dlouho, dokud nejsou objeveny všechny uzly dosažitelné z výchozího uzlu. Pokud ještě nějaké neprozkoumané vrcholy zbývají, zvolí se nový výchozí uzel. Takto se pokračuje, dokud nejsou prozkoumány všechny vrcholy.

Označení uzlů. Algoritmus rozděluje uzly do tří skupin:

- nové (zatím neobjevené),
- otevřené,
- uzavřené.

Na začátku jsou všechny vrcholy označeny jako nové ($stav[v] == -1$). Otevřenými se stanou po svém objevení ($stav[v] == 0$) a uzavřenými po kompletním prozkoumání svých sousedů ($stav[v] == 1$).

Při provádění algoritmu jsou každému vrcholu v přiřazeny značky $d[v]$ a $f[v]$, které odpovídají okamžiku otevření a uzavření uzlu. S pomocí těchto značek lze provádět další grafové algoritmy. Při objevení nového uzlu u je do vektoru p uložen jeho předchůdce v.

Algoritmus 4.3 - Prohledávání do hloubky

Inicializace

```
1. for ∀v ∈ V : stav[v] = −1 a p[v] = −1. Čítač značek i = 0
2. for ∀v ∈ V
3.   if stav[v] == −1
4.     DFS(v)
```

```
DFS(v)
1. stav[v] = 0
2. i = i + 1; d[v] = i
3. for ∀u ∈ Adj[v]
4.   if stav[u] == −1
5.     p[u] = v
6.     DFS(u)
7. stav[v] = 1
8. i = i + 1; f[v] = i
```

Časová složitost. Algoritmus je realizován rekurzivním způsobem a jeho složitost je $O(m+n)$, kde m je počet vrcholů a n je počet hran grafu. (KOLÁŘ, 2004)

4.6 Douglas-Peucker algoritmus

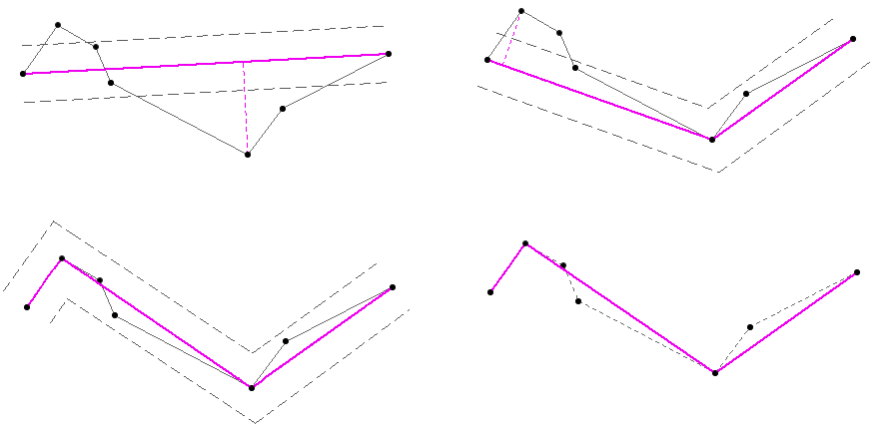
V průběhu agregace je třeba zjednodušit tvar redukovaného skeletonu agregovaného polygonu. Pro generalizaci bude použit Douglas-Peuckerův algoritmus, který patří mezi nejčastěji používané algoritmy pro generalizaci polylinií.

Algoritmus byl nezávisle na sobě publikován v roce 1973 dvěma autory - D. Douglasem a T. Peuckerem, bývá proto označován jako Douglas-Peucker algoritmus. V informatice je známý spíše jako Ramer algoritmus. Patří mezi globální algoritmy, které zohledňují tvar generalizované linie jako celku. Jedná se o rekurzivní algoritmus, který postupně přidává k linii vrcholy splňující danou podmínku.

Princip algoritmu. Princip ilustruje obr. 4.12 a alg. 4.4. Lomená čára $L = \{p_1, p_2, \ldots, p_n\}$ je nahrazena úsečkou $L' = p_1 p_n$ spojující počáteční a koncový bod původní čáry. Je definován koridor o šířce h. Pak je hledán bod p vně koridoru takový, že platí

$$d(p, L') = max(d(p_i, L'))$$

Pokud takový bod neexistuje, algoritmus končí. V opačném případě je bod p přidán k L' a pokračuje se hledáním nového bodu p. Vstupními parametry algoritmu jsou tak indexy počátečního a koncového bodu generalizované linie a šířka koridoru h.



Obr. 4.12: Princip Douglas-Peucker algoritmu

Algoritmus 4.4 - Douglas-Peucker(L, h, $start$, end)

```
1. if (end > start + 1) do
2.    max = start + 1
3.    d_max = dist(p_max, p_start p_end)
4.    i = max + 1
5.    do untill (i < end)
6.       d = dist(p_i, p_start p_end)
7.       if (d > d_max) do
8.          max = i; d_max = d
9.       i = i + 1
10.   if (d_max > h)
11.      L' ← p_max
12.      Douglas-Peucker(L,h,start,max)
13.      Douglas-Peucker(L,h,max,end)
```

5 STANOVENÍ GEOMETRICKÝCH A KARTOGRAFICKÝCH PODMÍNEK PRO AGREGACI

V této části, před představením vlastního návrhu algoritmu, budou stanoveny podmínky pro agregaci. Generalizační algoritmus byl navrhován tak, aby co nejlépe splnil poměrně široké požadavky kartografického i geometrického charakteru. Uveďme nejvýznamnější kartografické požadavky kladené na navrhované řešení:

- *kartografická věrnost výstupu,*
- *alespoň rámcové zachování plochy generalizovaných objektů,*
- *univerzalita (použitelnost pro různé tvary budov včetně moderní architektury),*
- *malý posunu těžiště agregovaného objektu vzhledem ke vstupním datům,*
- *zjednodušení tvaru při současném zachování výše uvedených parametrů.*

Z hlediska informatického požadujeme:

- *snadnou implementaci algoritmu,*
- *rozumnou výpočetní složitost,*
- *uchování topologie dat,*
- *schopnost předcházení tzv. self-intersections, které negativně ovlivňují kvalitu výstupů.*

Řada stanovených požadavků má poměrně protichůdný charakter, a nebude zřejmě jednoduché je beze zbytku při návrhu algoritmu splnit. Algoritmus využívá poměrně široké škály pomocných algoritmů a datových struktur, jejichž zkombinování nebude při implementaci snadné. Zachování polohy, plochy a topologie dat by měly být silnějšími stránkami algoritmu. S tím související kartografická věrnost výstupu (především zachování pravoúhlosti a orientace budov) je v teoretickém návrhu splnitelná, její cenou je však složitější implementace.

Ve zprávě AGENT (DA2, 2001) jsou stanoveny omezení pro generalizaci budov a shluků budov. Některé z nich, podpořené např. prací STEINIGERA a TAILLANDIERA i vlastním studováním českých mapových děl (Základní mapy, Topografické mapy) jsou dále uvedeny v jednotlivých podkapitolách, neboť definují podmínky kartografické přirozenosti a čitelnosti výstupů.

5.1 Kartograficky přirozené výstupy

Jedním z hlavních požadavků na algoritmus je kartografická věrnost výstupu. Tento aspekt se promítne nejen do samotného procesu agregace, ale také už při výběru prvků, které budou do agregace vstupovat.

Omezení liniovými prvky. Budovy a shluky budov v rámci sídelních systémů jsou omezeny především cestní sítí. Agregace by proto neměla spojovat budovy, které náleží do různých bloků. Tato skutečnost je stanovena jako jedna ze základních omezujících podmínek pro navrhovaný algoritmus. Obdobně by bylo možné stanovit podmínku, aby se neagregovaly budovy, mezi kterými leží například vodní tok, nebo které jsou od sebe výškově příliš vzdáleny. Tyto skutečnosti v algoritmu zahrnuty nejsou, jejich zabudování by však nebylo příliš náročné. Pro každou dvojici agregovaných budov můžeme zjišťovat, zda spojnice jejich centroidů kříží vodní tok nebo stanovený počet vrstevnic (viz obr. 5.1). Odpovídající vrstvy bychom samozřejmě museli mít k dispozici.

Obr. 5.1: Omezení liniovými prvky: *(a)* křížení vrstevnic, *(b)* křížení vodního toku

Zachování geografického kontextu. V rámci zachování geografického kontextu by měly být z agregace vyloučeny osamocené malé budovy, které by v mapě menšího měřítka neměly zaniknout. V rámci kontextu by také měl být zachován charakteristický vzor budov. Problematika zachování charakteristického vzoru byla rozvedena v části 3.2. V oblastech rozvolněnější zástavby by vůbec nemělo docházet k agregaci, ale spíše k typifikaci budov.

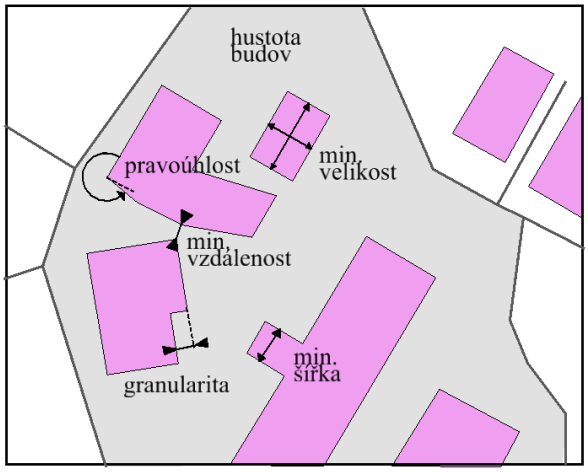
Identifikovat tyto oblasti by bylo vhodné před samotnou aplikací navrženého algoritmu. Jedná se o samostatný problém, řešený např. pomocí přístupů založených na gestalt teorii (STEINIGER a TAILLANDIER; LI et al., 2004) nebo kombinací bufferu a Voronoi diagramu (BASARANER a SELCUK, 2004).

Navržený algoritmus v této otázce řeší pouze pro české prostředí častý fenomén chatových oblastí. Budovy v chatových oblastech mají typicky čtvercový půdorys s malou plochou. Podle informací od ČÚZK se například vektorizace ZABAGED na základě ZM 10 řídila následujícím pravidlem. Mohly se uměle zvětšovat budovy právě v chatových oblastech tak, aby splňovaly kritérium minimální plochy $50m^2$, resp. se mu přiblížily (odpovídají čtverci o straně 7 metrů). Jedná se o postup v kartografii poměrně častý, tzv. kresba přes míru. V případě jiných datových sad mohou být minimální rozměry jiné, měly by však být podřízeny podmínkám čitelnosti, viz obr. 5.2.

Navržený algoritmus při načítání budov ze zdrojových souborů odfiltruje budovy menší než kritická hodnota S_{min}, v případě testování na datech ZABAGED bude S_{min} stanoveno na hodnotu $50m^2$.

4:1	1:1	min. velikost	min. tloušťka linie	min. vzdálenost	poznámky
▪	·	0.35 mm			plný čtverec
▬	-	0.30 mm		0.20 mm	úzký obdélník
				0.25 mm	prázdný prostor mezi prvky
				0.15 mm	podlouhlé budovy
□	-	0.50 mm	0.08 mm		prázdný čtverec

Obr. 5.2: Grafické limity (upraveno podle: AGENT, DA2, 2001, s. 32)



Obr. 5.3: Omezující podmínky pro budovy (upraveno podle: STEINIGER a TAILLANDIER, s. 2)

5.2 Zachování plochy a těžiště agregovaných oblastí

V souvislosti se zachováním kartografické věrohodnosti je třeba také minimalizovat změny geometrických vlastností budov. Mezi hlavní kritéria patří (podle AGENT, DA2, 2001) zachování přesnosti původního umístění, zachování absolutní a relativní orientace, s čímž souvisí i zachování tvaru - protáhlost budov, pravoúhlost, minimální velikost hran a výstupků budov (obr. 5.3).

Při agregaci musí vždy zákonitě dojít ke zvětšení plochy objektů, stejně tak k posunu těžiště. Při hodnocení algoritmu budou tyto charakteristiky vypočteny a posouzeny.

5.3 Různé typy budov, zjednodušení tvaru

Schopnost zpracování budov obecných tvarů představuje značně složitý problém. Agregace víceméně pravoúhlých budov bez zvláštních výstupků je poměrně jednoduchá, dobré výsledky v tomto případě dává algoritmus REGNAULDA, 1998 (viz část 3.3.1), větší problém však má v případě budov složitějších tvarů. Důležitou roli hraje nejen tvar jedné budovy, ale i tvar a poloha budov sousedních, které jsou adepty pro agregaci.

Algoritmus je proto navrhován se snahou provést agregaci bez explicitního zjišťování charakteristik jednotlivých budov (protáhlost, pravoúhlost, počet vrcholů, úhel natočení, minimální šířka...) tak, aby dokázal zpracovat budovy složitějších tvarů. Při finálním zjednodušení prvků je nutné brát v úvahu geometrické limity tvarů budov.

5.4 Změna datové reprezentace

Rastrová reprezentace dat, která diskrétně rozděluje prostor, má své výhody pro mnohé prostorové analýzy. Uveďme například široké využití mapové algebry, interpolace, filtrace nebo modelování geografických jevů. Také v ní není nutné řešit topologickou správnost objektů v takové míře, jako u vektorových dat. Naproti tomu vektorová reprezentace dat je paměťově méně náročná, umožňuje přímé propojení s databází, snadnou editaci, lepší provádění některých analýz (například síťové analýzy).

Databáze, z kterých vznikají státní mapová díla, turistické mapy, mapy na webových portálech atd. jsou uloženy ve vektorovém formátu. Algoritmy pro kartografickou generalizaci by proto měly umět pracovat s tímto typem dat. Jak bylo uvedeno v části 3.3.2, konverze mezi datovými reprezentacemi jsou náročné. Cílem je proto navrhnout nový algoritmus pracující pouze ve vektorovém prostředí.

6 NÁVRH GENERALIZAČNÍHO ALGORITMU

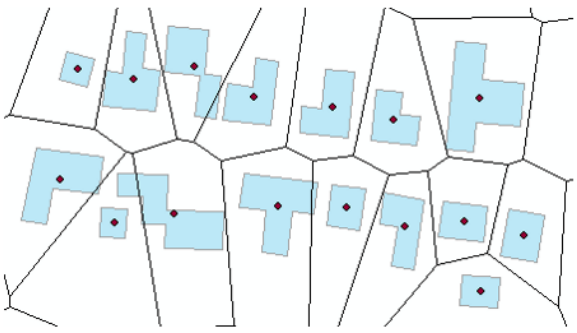
V této části je představen vlastní návrh generalizačního algoritmu. Problém generalizace technikou agregace je netriviální, při návrhu je nutné zohlednit řadu geometrických i kartografických požadavků, které jsou v mnohých případech protichůdné. Cílem je navrhnout algoritmus s co největší mírou automatizace tak, aby se minimalizoval počet ručních korekcí výsledků.

Algoritmus bude vhodné aplikovat na datové sady, ve kterých jsou budovy znázorněny jako samostné objekty, případně jako bloky budov. Tuto podmínku splňují datové sady do měřítka podrobnosti 1:100 000. Vstupní datové sady by měly být před vstupem do algoritmu generalizovány tak, aby neobsahovaly nadbytečné vrcholy. Vstupním i výstupním formátem pro generalizaci je shapefile. Uživatelským vstupem algoritmu bude navíc hodnota minimální plochy S_{min} (viz část 5.1) a měřítkové číslo M měřítka mapy, do které chceme generalizovat.

Průběh navrhovaného algoritmu můžeme v zásadě rozdělit do dvou kroků: určení budov vhodných pro agregaci a vlastní postup agregace. První krok zahrnuje vyhledání sousedících budov, zhodnocení kritérií pro jejich agregaci a určení skupin budov, které budou tvořit agregovaný polygon. Druhý krok zahrnuje vytvoření redukovaného skeletonu agregovaného polygonu a jeho následné rozšíření.

6.1 Určení budov vhodných pro agregaci

Vyhledání sousedících budov. Definujme navzájem sousedící budovy na základě sousednosti Voronoi buněk. Generátory Voronoi diagramu tvoří centroidy generalizovaných budov, viz obr. 6.1.



Obr. 6.1: Voronoi diagram nad centroidy budov pro určení sousednosti polygonů

Zhodnocení kritérií pro agregaci. Označme dvě navzájem sousedící budovy jako P, Q. Základním kritériem pro rozhodnutí, zda budou sloučeny, je hodnota jejich minimální vzdálenosti $d(P, Q)$, vypočtená pomocí algoritmu z části 4.2. Kritická hodnota vzdálenosti d_{max} bude stanovena v závislosti na uživatelském vstupu měřítkového čísla M - bude odpovídat vzdálenosti $0,25mm$ ve výsledném měřítku (hodnota stanovená dle grafických limitů, viz. obr. 5.2):

$$d_{max} = 0,25 \cdot \frac{M}{1000}$$

Agregovat se tedy budou pouze takové polygony P, Q, pro které platí:

$$d(P, Q) < d_{max}$$

V souladu s částí 5.1 jsou ze zpracování vynechány polygony P, pro které platí:

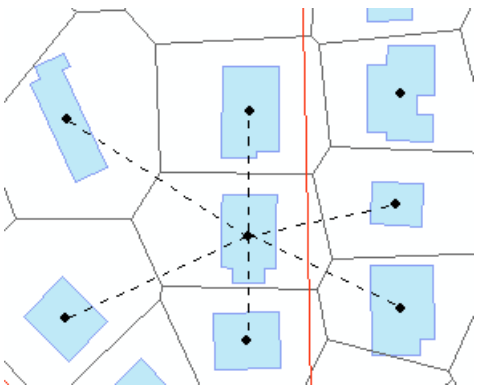
$$S(P) < S_{min}$$

Pro testování algoritmu na datech ZABAGED je $S_{min} = 50m^2$. Hodnota je stanovena na základě textu v části 5.1.

Množina omezení. Definujme množinu omezení O tvořenou liniovými segmenty představující prvky, přes které nesmí být agregace prováděna (silnice, vodní toky, vrstevnice...) - viz část 5.1. Následně testujme vzájemnou polohu spojnic centroidů agregovaných budov s prvky množiny O, viz obr. 6.2. K agregaci dojde pouze v případě, je-li splněna podmínka

$$O \cap s_c = \emptyset$$

kde s_c je testovaná spojnice centroidů.



Obr. 6.2: Testování vzájemné polohy spojnic centroidů a segmentů omezení

Uveďme vztahy pro výpočet souřadnic centroidu polygonu:

$$C_x = \frac{1}{6A}\sum_{i=0}^{n-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \tag{6.1}$$

$$C_y = \frac{1}{6A}\sum_{i=0}^{n-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \tag{6.2}$$

kde A je plocha polygonu:

$$A = \frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)$$

n je počet vrcholů polygonu, poslední vrchol je shodný s počátečním.

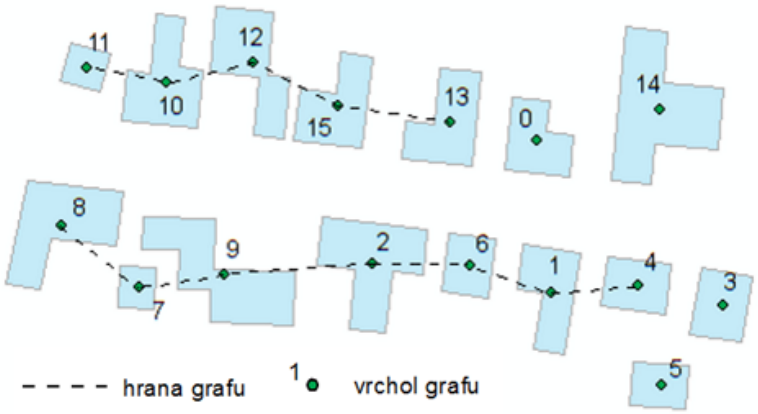
Označme P seznam všech vstupních polygonů, P_{agr} seznam všech polygonů, které budou agregovány, $C(P)$ seznam všech centroidů a P_{adj} seznam sousedů polygonu $P_i \in P$. Platí, že P_{agr} je podmnožinou P. Vlastní algoritmus pro nalezení budov vhodných pro agregaci lze zapsat takto:

Algoritmus 6.1 - Určení budov pro agregaci

```
Vstup: seznam P, výstup: P_adj for ∀P_i ∈ P
1. for ∀P_i ∈ P
2.   if (S(P_i) > S_min)
3.     P_agr ← P_i
4. for ∀P_i ∈ P_agr
5.   urči centroid C(P_i) dle vztahů 6.1, 6.2
6.   Voronoi diagram V(C(P)) ← C(P_i)
7. for ∀ buňka f_p ∈ V(C(P))
8.   for ∀ hrana h ∈ f_p
9.     urči sousedící polygon s
10.    if (O ∩ s_c = ∅ and vzdálenost d(s,p) < d_max)
11.      P_adj ← s
```

Vytvoření seznamu sousedů. Sekvenčním procházením Voronoi buněk jsou každému polygonu přiřazeny indexy sousedních polygonů, s nimiž se má agregovat (za dodržení podmínek minimální plochy a průsečíků s omezeními). Tím se vytvoří pro každý polygon lineární seznam sousedů pro agregaci, P_{adj} (alg 6.1, krok 7-11).

Množina všech lineárních seznamů P_{adj} je spojovou reprezentací grafu G, představujícího skupiny budov pro agregaci (příklad viz. obr. 6.3). Každý podstrom grafu G definuje jednu skupinu budov pro agregaci. Určení - explicitní vyjádření - těchto podstromů je cílem první části algoritmu. Pro jejich nalezení je na graf G spuštěn algoritmus prohledávání do hloubky (alg. 4.3).



Obr. 6.3: Graf představující skupiny budov pro agregaci

Spojová reprezentace grafu G (tj. množina $\forall P_{adj}$) vypadá pro obr. 6.3 následovně:

$0 \rightarrow \emptyset$	$6 \rightarrow 2, 1$	$12 \rightarrow 10, 15$
$1 \rightarrow 4, 6$	$7 \rightarrow 9, 8$	$13 \rightarrow \emptyset$
$2 \rightarrow 6, 9$	$8 \rightarrow 7$	$14 \rightarrow \emptyset$
$3 \rightarrow \emptyset$	$9 \rightarrow 7, 2$	$15 \rightarrow 12$
$4 \rightarrow 1$	$10 \rightarrow 12, 11$	
$5 \rightarrow \emptyset$	$11 \rightarrow 10$	

Nalezení podstromů grafu. Prohledávání do hloubky grafu G je zahájeno ve vrcholu v (pro ilustraci např. vrchol 0 na obr. 6.3). Jedná se o izolovaný vrchol, prohledávání proto pokračuje dalším vrcholem (např. 1 na obr. 6.3). Po objevení všech vrcholů dosažitelných z vrcholu 1 (tj. po objevení celého podstromu) se zvolí nový výchozí vrchol. V algoritmu se pokračuje tak dlouho, dokud není graf kompletně prohledán.

Pokud v průběhu DFS ukládáme vždy do jednoho nového lineárního seznamu sekvenci vrcholů tvořících dílčí podstrom (při současném ignorování izolovaných vrcholů), je výsledkem prohledávání množina lineárních seznamů představujících podstromy grafu G, pro obr. 6.3 ve tvaru:

$$8, 7, 9, 2, 6, 1, 4$$
$$11, 10, 12, 15, 13$$

Každý z lineárních seznamů tedy představuje jeden podstrom, tj. jednu skupinu budov, které se budou vzájemně agregovat.

6.2 Vlastní agregace

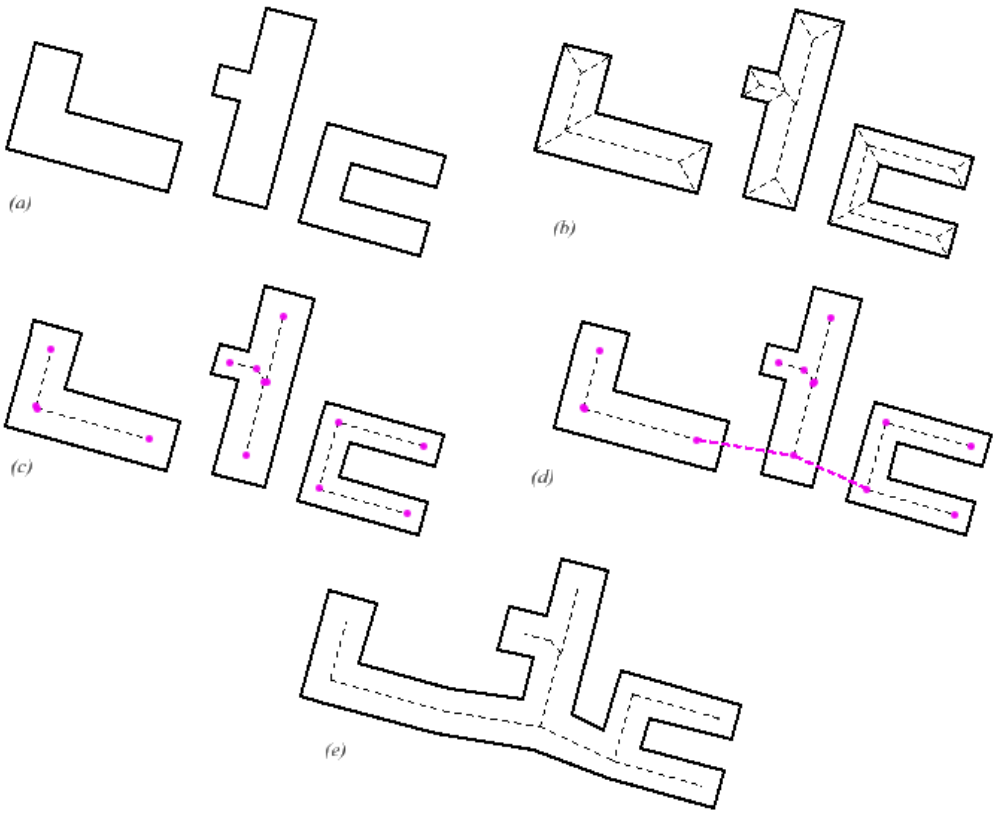
Pro vlastní agregaci byl navržen přístup s využitím konstrukce topologické kostry, konkrétně straight skeletonu, konstruovaném nad agregovanými polygony. Na úvod popišme a ilustrujme funkcionalitu algoritmu na konkrétním případě (viz obr. 6.4).

Obr 6.4a znázorňuje budovy určené pro agregaci. V první fázi zkonstruujme pro každou z budov straight skeleton (obr. 6.4b). Z každého straight skeletonu získejme redukovaný straight skeleton (obr. 6.4c). Následně agregujme redukované straight skeletony Kruskalovým algoritmem. Úloha agregace budov je tedy převedena na úlohu agregace podkoster grafu (obr. 6.4d).

Posledním krokem celého algoritmu je rekonstrukce polygonu z vytvořené kostry (obr. 6.4e). Vytvořme z kostry pohybem jejích vrcholů po bisektorech úhlů směrem ven malý polygon, jehož plocha se limitně blíží 0, a nazvěme ho první aproximací agregovaného polygonu. Poté rozšiřujme tento pomocný polygon (opět po bisektorech úhlů), dokud

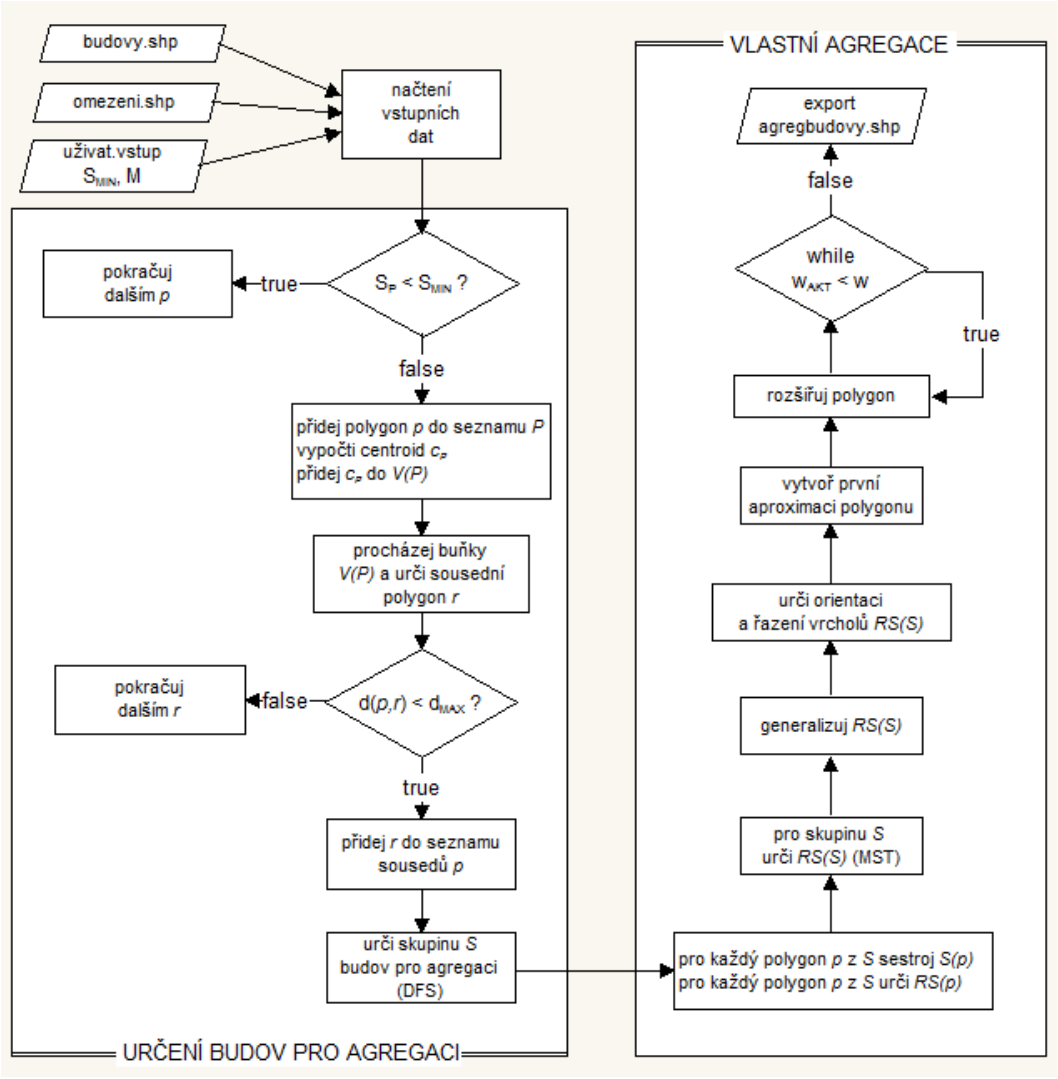
$$w_{akt} \leq w$$

kde w_{akt} je aktuální šířka polygonu, w je ukončovací šířka. Nastavení hodnoty w bude diskutováno v části 6.2.3.



Obr. 6.4: Princip agregace polygonů

V následujících podkapitolách jsou podrobněji popsány jednotlivé kroky vlastní agregace polygonů - první se věnuje tvorbě a generalizaci redukovaného skeletonu agregovaného polygonu, druhá rozšíření redukovaného skeletonu na první aproximaci polygonu a třetí vlastnímu rozšiřování polygonu. Funkcionalita celého algoritmu je znázorněna na obr. 6.5.



Obr. 6.5: Schéma generalizačního algoritmu

6.2.1 Tvorba a generalizace redukovaného straight skeletonu agregovaného polygonu

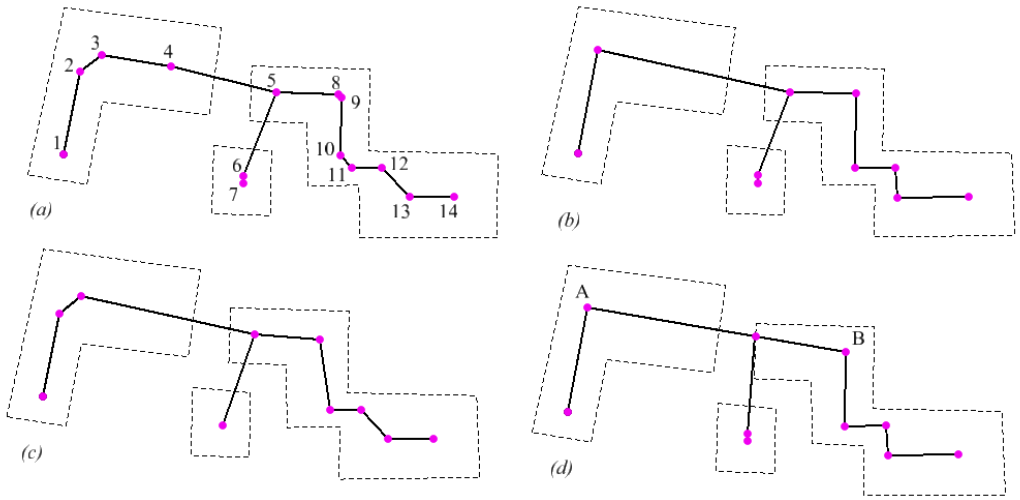
Tvorba redukovaného skeletonu agregovaného polygonu. Pro každý z polygonů určených pro agregaci sestrojme straight skeleton. Ze skeletonu získejme redukovaný skeleton (obr. 6.4b,c,d). Každý z redukovaných skeletonů považujme za graf $G_i = (V_i, E_i, \varepsilon_i)$, kde V_i a E_i jsou množiny vrcholů a hran redukovaných skeletonů. Nyní uvažujme, že každý z těchto grafů je podgrafem grafu $H = (V, E, \varepsilon)$. Graf H je tvořen všemi vrcholy a všemi hranami všech redukovaných skeletonů. Minimální kostra grafu H pak určuje redukovaný straight skeleton agregovaného polygonu. Ohodnocení hran představují jejich délky.

Generalizace redukovaného skeletonu. Vytvořený redukovaný skeleton je vhodné generalizovat - především odstranit krátké výstupky a zjednodušit jeho tvar. Definujme na tomto místě pojem větev.

Větev. Podgraf grafu G, který je stromem a pro každý z jeho uzlů platí:

$$s_u \leq 2$$

kde s_u je stupeň uzlu (tj. počet hran s uzlem incidujících), nazvěme větví grafu.



Obr. 6.6: Generalizace redukovaného skeletonu agregovaného polygonu

Na obr. 6.6a je znázorněn redukovaný straight skeleton agregovaného polygonu. Cílem je provést jeho generalizaci. Navržený algoritmus generalizuje redukovaný skeleton po větvích, tj. v případě na obr. 6.6 dojde ke generalizaci tří větví:

- 1-2-3-4-5
- 5-6-7
- 5-8-9-10-11-12-13-14

Generalizace je prováděna Douglas-Peuckerovým algoritmem (část 4.6). V případě, že kostra obsahuje krátké větve, jsou tyto větve zcela odstraněny. Douglas-Peucker algoritmus požaduje jako vstup šířku koridoru h. Pro odstranění větví je třeba stanovit minimální délku větve h_{min}. Hodnoty h a h_{min} budou určeny v závislosti na uživatelském vstupu měřítkového čísla M - budou odpovídat kritické hodnotě $0,2mm$ ve výsledném měřítku:

$$h = h_{min} = 0,2 \cdot \frac{M}{1000} \tag{6.3}$$

Hodnota $0,2mm$ je stanovena na základě grafických limitů (obr. 5.2). Stejnou hodnotu použili (pro šířku pásu h Douglas-Peuckerova algoritmu) ve své práci STEINIGER a TAILLANDIER.

Zhodnocení kvality generalizace. Zdůrazněme, že kvalita generalizace redukovaného skeletonu bude ve výsledku výrazně ovlivňovat tvar polygonu vzniklého z této kostry. Z implementačního hlediska se jedná o netriviální úkol a navržený algoritmus je v otázce generalizace redukovaného skeletonu poněkud omezen. Na obr. 6.6 ilustrujme „ideální“, „střední“ a „reálný“ stav generalizované kostry.

Situace na obr. 6.6b, 6.6c a 6.6d znázorňují různou míru generalizace redukovaného skeletonu. Obr. 6.6d představuje „ideální stav“, kdy:

- je zachována hlavní linie, podél které jsou budovy umístěny (spojnice AB),
- jsou odstraněny hrany, které vznikly v místech původních pravých úhlů (hrana 2-3, 8-9, 10-11, 12-13) tak, aby tyto úhly zůstaly po rekonstrukci zachovány,
- jsou odstraněny krátké segmenty,
- jsou odstraněny nadbytečné vrcholy v jednotlivých větvích.

Obr. 6.6b představuje „střední stav“, kdy:

- jsou odstraněny hrany, které vznikly v místech původních pravých úhlů (hrana 2-3, 8-9, 10-11, 12-13) tak, aby tyto úhly zůstaly po rekonstrukci zachovány,
- jsou odstraněny krátké segmenty,
- jsou odstraněny nadbytečné vrcholy v jednotlivých větvích.

Obr. 6.6c představuje „reálný stav“, který bude dosažen aplikací navrženého algoritmu:

- jsou odstraněny krátké segmenty,
- jsou odstraněny nadbytečné vrcholy v jednotlivých větvích.

Algoritmus dosahuje pouze stavu ilustrovaného na obr. 6.6c z následujících důvodů:

- Douglas-Peucker algoritmus zjednodušuje linii pouze odstraněním bodů, nemění jejich polohu,
- generalizace probíhá po větvích, takže lze těžko zachovat hlavní linii (AB na obr. 6.6d),
- nahrazení hran v místě pravých úhlů by vyžadovalo identifikaci těchto hran a jejich správné nahrazení.

Pro další rozpracování navrženého algoritmu představuje krok generalizace redukovaného skeletonu jednu z největších výzev.

Nalezení větví redukovaného skeletonu. Uveďme zde ještě algoritmus pro nalezení větví redukovaného skeletonu. Redukovaný skeleton je uložen ve formě spojové reprezentace grafu. Pro nalezení větví je využito prohledávání do hloubky. Větve jsou ukládány v kontejneru `branches`. Počet větví $RS(P)$ je určen následujícím vztahem:

$$z = \left(\sum_{i=0}^{n} s_u - 1 \right) + 1$$

kde n je počet uzlových vrcholů, s_u je stupeň uzlu.

Algoritmus 6.2 - Nalezení větví RS(P)

Vstup: redukovaný skeleton agregovaného polygonu $RS(P)$, výstup: dvojrozměrný vektor branches

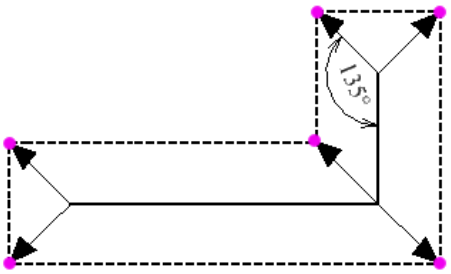
```
1. urči počet větví RS(P) z
2. číslo větve branchNo = 0
3.   if (z ≠ 1)
4.     for ∀ vrchol u ∈ RS(P)
5.       if (stav[u] == −1 and s_u > 2)
6.         aktuální uzel nodeDFS = u
7.         DFS(u)
8.   if (z == 1)
9.     najdi uzel u takový, že s_u == 1
10.    DFS2(u) - klasické prohledávání do hloubky, alg. 4.3

DFS(u)
1. stav[u] = 0
2. for ∀ vrchol v ∈ Adj[u]
3.   if (s_v > 2)
4.     if (stav[v] == −1)
5.       branches[branchNo].pushback(v)
6.       branches[branchNo].pushfront(nodeDFS)
7.       branchNo++
```

```
8.   else if (s_v == 1)
9.        branches[branchNo].pushback(v)
10.       branches[branchNo].pushfront(nodeDFS)
11.       branchNo++
12.  else
13.    if (stav[v] == −1)
14.       branches[branchNo].pushback(v)
15.       DFS(v)
```

6.2.2 První aproximace agregovaného polygonu

Vezměme generalizovaný redukovaný skeleton agregovaného polygonu a rozšiřme ho pohybem po bisektorech úhlů směrem ven (viz obr. 6.7). Z 1D entity tvořené stromem tak vznikne 2D entita představující polygon. Nazvěme ji první aproximací polygonu.



Obr. 6.7: První aproximace polygonu

První aproximace polygonu musí splňovat dvě podmínky:

- musí se jednat o jednoduchý polygon (bez vnitřních intersekcí),
- polygon musí být správně orientován (pro řešení práce je zvolena counter-clockwise orientace).

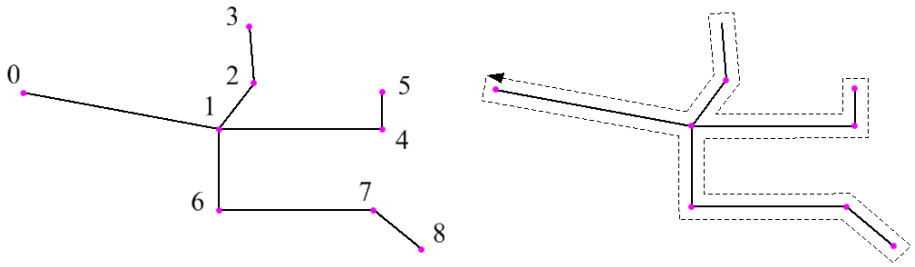
Korektní konstrukce první aproximace polygonu je zajištěna dvěma kroky. Nejprve určeme správnou orientaci polygonu. K tomu využijme modifikovaného prohledávání do hloubky. Poté proveďme vlastní změnu prostorové dimenze.

Určení správné orientace vrcholů polygonu. Ilustrujme postup na obr. 6.8 a 6.9. Mějme k dispozici redukovaný straight skeleton ve formě spojové reprezentace grafu, jeho grafická reprezentace viz obr. 6.8 vlevo. Určeme správnou orientaci a řazení vrcholů pro tento graf tak, aby z něho bylo možno prostým pohybem po bisektorech vytvořit polygon v counter-clockwise orientaci. Řazení pro obr. 6.8 by bylo následující: $0-1-6-7-8-7-6-1-4-5-4-1-2-3-2-1-0$. Vlastní modifikované prohledávání do hloubky ilustruje obr. 6.9.

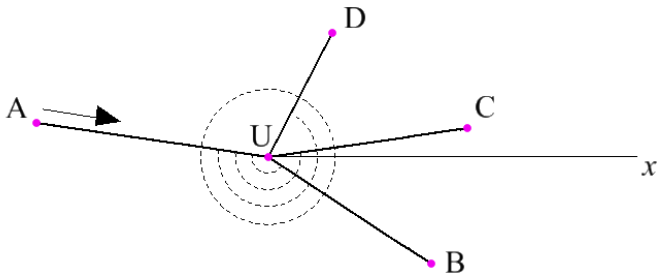
Vstupme do algoritmu pro prohledávání do hloubky v koncovém bodě grafu (bod A). Dojdeme-li do uzlového bodu (bod U), postupujme následovně. Označme úhly

$$\alpha = \angle AUB, \beta = \angle AUC, \gamma = \angle AUD, \delta = \angle AUA$$

Uložme úhly podle jejich velikosti do prioritní fronty PQ. Vyzvedávejme postupně z PQ úhly od nejmenšího po největší a procházejme odpovídající segmenty kostry, dokud není graf kompletně prohledán.



Obr. 6.8: Vytvoření první aproximace polygonu



Obr. 6.9: Určení úhlů pro správnou orientaci vrcholů kostry

Změna prostorové dimenze. Určeme pro každý vrchol redukovaného skeletonu osu úhlu incidujících hran. V případě koncových vrcholů je bisektor určen jako úhel incidující hrany $\mp 135°$ (viz obr. 6.7). Při counter-clockwise orientaci jsou bisektory vždy orientovány směrem vpravo od redukovaného skeletonu.

Vytvořme první aproximaci polygonu ve vzdálenosti d od redukovaného skeletonu. Souřadnice vrcholů se změní podle vztahů:

$$p.x = p.x + \cos(b_p) \cdot \frac{d}{\sin(a_p)} \tag{6.4}$$

$$p.y = p.y + \sin(b_p) \cdot \frac{d}{\sin(a_p)} \tag{6.5}$$

kde b_p je úhel bisektoru vrcholu p, $\alpha_p \in \langle 0; \frac{\pi}{2} \rangle$ je polovina úhlu sevřeného incidujícími hranami vrcholu p.

Hodnota vzdálenosti d musí být velmi malá, aby při tvorbě první aproximace polygonu nedošlo k žádné topologické změně. Obecně nemůžeme zaručit, že ke změně topologie nedojde (viz následující část 6.2.3). Za předpokladu, že redukovaný skeleton je generalizován a nevyskytují se v něm body vzájemně velmi blízké, může být d např:

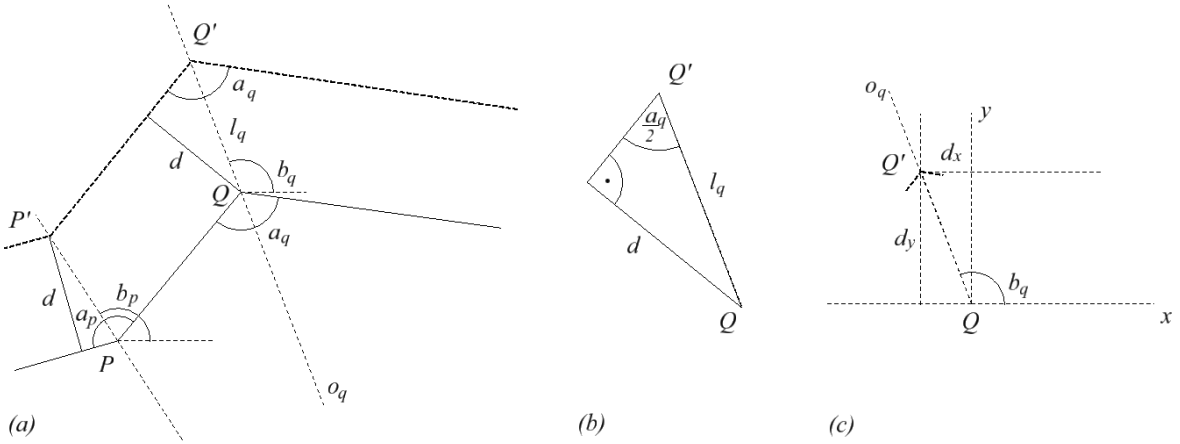
$$d = 0,001m$$

Tato hodnota je zvolena empiricky a je použita v implementaci algoritmu.

Vztahy 6.4, 6.5 lze odvodit pomocí obr. 6.10. Označme $Q[x, y]$ původní vrchol, $Q'[x', y']$ hledaný vrchol. Dle obr. 6.10a:

$$d = dist(PQ, P'Q'),$$

b_q představuje úhel bisektoru vrcholu Q, $a_q \in \langle 0, \pi \rangle$ úhel sevřený hranami incidujícími s vrcholem Q.



Obr. 6.10: Odvození vztahů pro posun vrcholů po bisektorech

Souřadnice vrcholu Q' lze vypočítat jako:

$$Q' = Q + \overrightarrow{QQ'}$$

Klíčovým úkolem je tedy určit složky u_x,u_y vektoru $\overrightarrow{QQ'}$. Z obr. 6.10b plyne:

$$l_q = \frac{d}{\sin \frac{a_q}{2}}$$

Dle obrázku 6.10c můžeme psát:

$$u_x = d_x = \cos(b_q) \cdot l_q$$
$$u_y = d_y = \sin(b_q) \cdot l_q$$

a tedy

$$x' = x + u_x$$
$$y' = y + u_y$$

6.2.3 Zpětná rekonstrukce polygonu

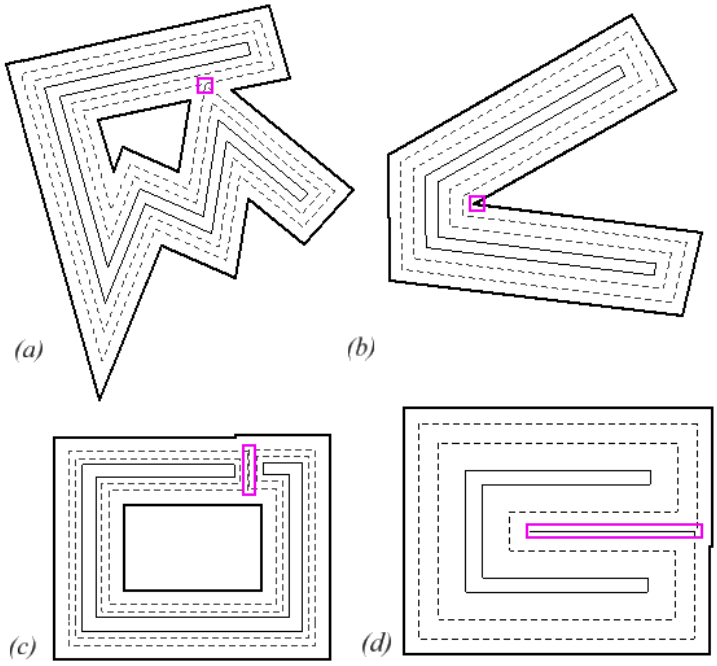
Označme první aproximaci polygonu P jako $P_A = \{p_1, p_2, \ldots, p_n\}$, kde $p_i = [x_i, y_i]$ představuje vrchol P_A. Rozšiřujme P_A posunováním vrcholů p_i po osách úhlů incidujících hran směrem ven, dokud nedosáhneme požadované šířky w. Jedná se o postup inverzní ke konstrukci straight skeletonu, nazvěme ho *zpětnou rekonstrukcí polygonu P*. Podobně jako při konstrukci skeletonu při něm dochází k několika typům událostí (topologickým změnám), viz obr. 6.11:

- **událost vertex-vertex** - obr. 6.11b), dochází k zániku hrany
- **událost vertex-hrana** - obr. 6.11a), dochází ke střetu nekonvexního vrcholu a protilehlé hrany, polygon je rozdělen na dvě části. Podobně může dojít ke střetu dvou nekonvexních vrcholů proti sobě, je to v podstatě hraniční situace střetu vertexu a hrany
- **událost hrana-hrana** - obr. 6.11c,d) - dojde ke střetu rovnoběžných hran, důsledkem je buď vznik díry nebo jednoduchého polygonu

Princip konstrukce. Pro správnou konstrukci nového polygonu je nutné uvedené události správně detekovat (algoritmus 6.3). Pro detekci je použit stejný princip jako využili FELKEL a OBDRŽÁLEK, 1998. Pro každý vertex je určen nejbližší průsečík:

- buď jako průsečík s bisektory sousedících vertexů,
- nebo jako průsečík vertexu a hrany.

Průsečík vertexu a hrany je určován stejným způsobem jako při konstrukci straight skeletonu pro nekonvexní polygony (část 4.3). Je také využita datová struktura SLAV. Průsečíky I jsou stejně jako u Felkela a Obdržálka ukládány v prioritní frontě PQ a zpracovávány podle vzdálenosti d od generující hrany (alg. 6.3). Algoritmus 6.4 popisuje zpracování detekovaných událostí. Označme vrcholy generující I jako v_{LEFT}, v_{RIGHT}. V případě průsečíku vzniklého jako událost vertex-hrana $v_{LEFT} = v_{RIGHT}$.



Obr. 6.11: Topologické změny polygonu při rekonstrukci polygonu. *(a)* střetnutí vertexu a protější hrany - vznik díry, *(b)* střetnutí dvou vertexů - zánik hrany, *(c)* střetnutí dvou hran - vznik díry, *(d)* - střetnutí dvou hran bez vzniku díry

Algoritmus 6.3 - Identifikace událostí: určení průsečíků

1. for $\forall$ vrchol $v_i \in P$
2. spočti průsečíky I_1, I_2 bisektorů b_v, b_{prev} a b_v, b_{next}
3. urči vzdálenosti $d_1 = d(I_1, e_1 = v_i v_{prev})$ a $d_2 = d(I_2, e_2 = v_i v_{next})$
4. if $(b_v > \pi)$
5. urči průsečík B - viz. část 4.3
6. urči vzdálenost $d_3 = d(B, e)$, e je protilehlá hrana
7. urči $d_{min} \in \{d_1, d_2, d_3\}$
8. if $(d_{min} == d_1)$
9. $PQ \leftarrow I_1$, $I_1.type == convex$
10. else if $(d_{min} == d_2)$
11. $PQ \leftarrow I_2$, $I_2.type == convex$
12. else // $(d_{min} == d_3)$
13. $PQ \leftarrow I_3$, $I_3.type == nonconvex$

Algoritmus 6.4 - Zpracování událostí

1. while $(PQ$! $empty)$
2. $PQ \rightarrow I$
3. $PQ.pop$
4. if $(v_{LEFT}.done$ and $v_{RIGHT}.done)$
5. continue;
6. else

```
7.      if (I.type == convex)
8.        vytvoř vrchol U na místě I
9.        v_LEFT.done = true, v_RIGHT.done = true
10.       modifikuj LAV - viz část 4.3
11.       urči I_U - alg. 6.3
12.       PQ ← I_U
13.     if (I.type == nonconvex)
14.       vytvoř vrcholy U_1, U_2 na místě I
15.       v_LEFT.done = true, v_RIGHT.done = true
16.       modifikuj SLAV - viz část 4.3
17.       urči I_U1, I_U2 - alg. 6.3
18.       PQ ← I_U1, PQ ← I_U2
```

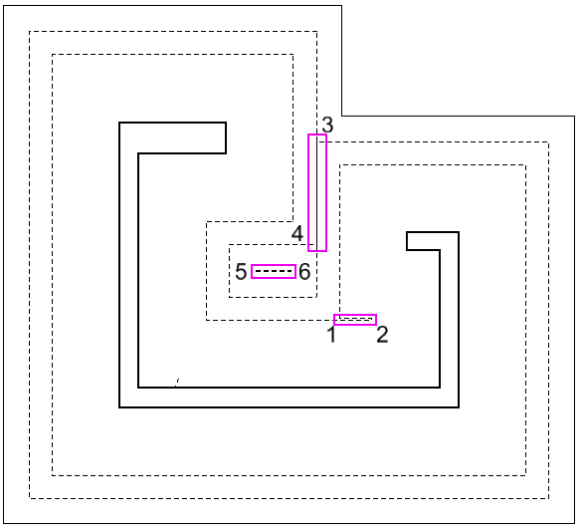
Objasněme ještě v dalším textu zpracování události hrana-hrana - bude zřejmé, že i tento typ události lze detekovat a zpracovat stejným způsobem jako události vertex-vertex a vertex-hrana.

Událost hrana-hrana. K této události může dojít v případě, kdy se spolu střetnou dvě rovnoběžné hrany. Událost hrana-hrana může být dvojího typu: buď při ní dochází zároveň i k zániku hrany (obr. 6.11d), nebo pouze ke vzniku díry (obr. 6.11c). Rozeberme nyní obě z těchto situací na příkladu obr. 6.12, události budou zpracovány v pořadí 1-4:

- událost 1 a 2: dojde k detekci dvou událostí ve stejné vzdálenosti od původního polygonu. Událost 1 je typu rozdělení (událost vertex-hrana), událost 2 je typu zánik (událost vertex-vertex). Jako první bude zpracována událost 1 - dojde k rozdělení LAV na dvě části. Jako druhá bude zpracována událost 2 - LAV tvořený vrcholy na místě průsečíků 1 a 2 bude mít nulovou plochu a pro výsledný polygon již nemá význam,
- událost 3 a 4: dojde k detekci dvou událostí ve stejné vzdálenosti, obě budou typu rozdělení (událost vertex-hrana). Při zpracování události 3 se LAV znovu rozpadne. Při zpracování události 4 vznikne další LAV, jeden z nich je tvořen pouze vrcholy na místě průsečíků 3 a 4, opět bude mít nulovou plochu a pro výsledný polygon již nemá význam.

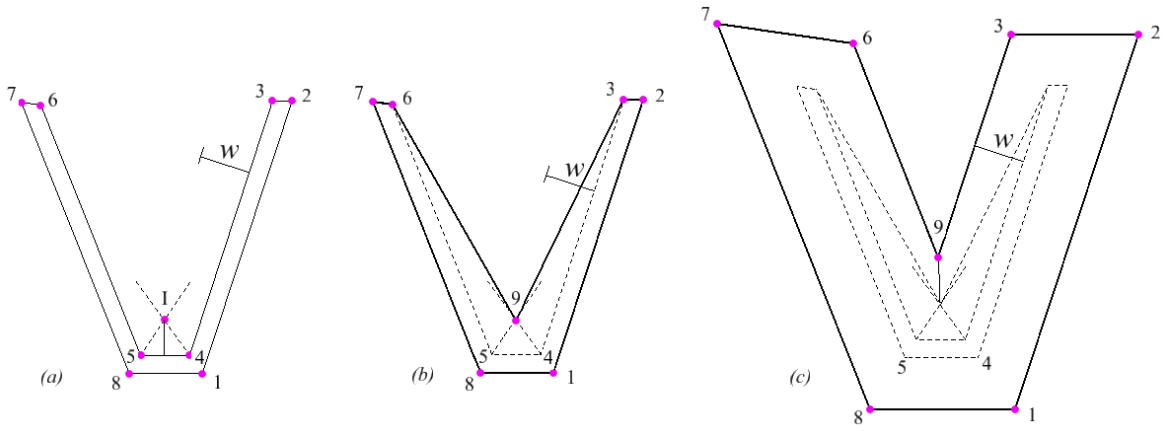
V předchozím textu byla objasněna detekce a zpracování událostí, k nimž dochází při zpětné rekonstrukci polygonu. V následujícím textu budou vyloženy změny, které byly do algoritmu Felkela a Obdržálka zabudovány, aby mohla být zpětná rekonstrukce úspěšně dokončena.

Připomeňme, že algoritmus Felkela a Obdržálka neaktualizuje souřadnice všech vrcholů polygonu P při zpracování události I. Dojde pouze k zapojení vrcholu V na místě průsečíku I do SLAV (viz obr. 6.13b). Po zpracování všech událostí je proto nutné aktualizovat souřadnice dosud nezpracovaných vrcholů rekonstruovaného polygonu - tím je získán finální rekonstruovaný polygon (obr. 6.13c).



Obr. 6.12: Střet rovnoběžných hran

Do datové struktury Felkela a Obdržálka byly proto přidány dva nové atributy pro vrcholy ve SLAV: `double Idist` a `int offset`. Navíc je přidána globální proměnná `int sizeOfOffset`, jejíž funkcionalita souvisí s atributem `offset`. Vyložme nejprve funkcionalitu atributu `Idist`.



Obr. 6.13: Zapojení vrcholu na místě průsečíku I do SLAV

Atribut Idist. Nechť P_A představuje první aproximaci polygonu P. Atribut $Idist$ udržuje pro každý vrchol $p_i \in P_A$ hodnotu:

$$p_i.Idist = d(p_i, P_A)$$

Před započetím rekonstrukce polygonu je tedy hodnota atributu pro každý vrchol $p_i \in P_A$ nulová.

Hodnota w (viz obr. 6.13) představuje ukončovací podmínku zpětné rekonstrukce polygonu. Označme vzdálenost $d(I, 45)$ na obr. 6.13a jako w_{akt} - představuje vzdálenost průsečíku I od generující hrany. Na obr. 6.13b je vrchol 9 na místě průsečíku I zapojen do LAV a je třeba mu nastavit hodnotu atributu $Idist$:

$$Idist = w_{akt}$$

Žádný další průsečík, pro který by platilo

$$w_{akt} \leq w \tag{6.6}$$

neexistuje. Zpracovávání průsečíků z prioritní fronty je ukončeno a dle algoritmu 6.5 dojde k výpočtu souřadnic výsledného rekonstruovaného skeletonu.

Atribut offset. Atribut *offset* udržuje pro každý vrchol v informaci, ke kterému LAV náleží. Příslušnost k LAV je využita při exportu rekonstruovaného polygonu do shapefilu, také je udržena informace o počtu LAV ve SLAV.

Před započetím zpracování je nastavena globální proměnná *sizeOfOffset* a každému vrcholu $p_i \in P_A$ nastavena hodnota atributu *offset*:

$$sizeOfOffset = 1$$
$$p.offset = sizeOfOffset$$

Dojde-li k události vedoucí k rozpadu LAV, je hodnota proměnné *sizeOfOffset* navýšena o 1. Označme jeden LAV jako $LAV1$, druhý jako $LAV2$. Pro každý vrchol $v \in LAV2$ aktualizujme hodnotu *offset*:

$$v.offset = sizeOfOffset$$

Algoritmus 6.5 - Finální aktualizace souřadnic rekonstruovaného polygonu

```
1. for ∀ vrchol p ∈ P
2.   if (! p.done)
3.     double l = w − p.Idist
4.     aktualizuj souřadnice vrcholu p podle vztahů 6.4, 6.5 (část 6.2.2), kde d = l
```

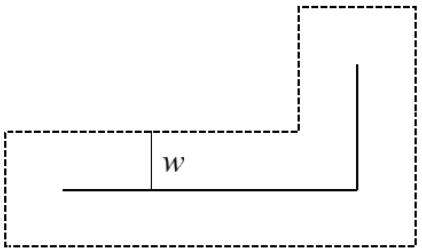
Poslední neurčenou hodnotou je ukončovací podmínka w zpětné rekonstrukce polygonu, která určuje vzdálenost rekonstruovaného polygonu P od první aproximace polygonu P_A.

Stanovení hodnoty w. Hodnota w je nezbytná pro ukončení běhu algoritmu. Měla by být volena s ohledem na data vstupující do agregačního algoritmu. Plocha a poloha agregovaného polygonu by se měly co nejvíce přibližovat charakteristikám původních polygonů. Současně by hodnota w měla být větší než kritéria čitelnosti pro dané měřítko.

Pro každý nový polygon P bude tedy hodnota w různá. V navrženém algoritmu byl zvolen poměrně jednoduchý způsob určení hodnoty w, je však docela efektivní:

$$w = \frac{\sum_{i=0}^{n} S_i}{\sum_{j=0}^{m} l_j} \cdot \frac{1}{2} \tag{6.7}$$

kde n je počet původních polygonů, S_i je plocha každého z polygonů, m je počet segmentů kostry agregovaného polygonu, l_j je délka každého segmentu.



Obr. 6.14: Stanovení hodnoty w

Koeficient $\frac{1}{2}$ je ve vztahu pro výpočet w z prostého důvodu - w určuje pouze polovinu šířky výsledného rekonstruovaného polygonu, viz obr. 6.14.

7 IMPLEMENTACE ALGORITMU

Algoritmus byl implementován pomocí volně dostupných knihoven *CGAL*, *Boost* a *ShapeLib* v prostředí Microsoft Visual C++ 2010 Express. Zde musí být uvedeno, že tento postup se mírně odlišuje od zadání diplomové práce. Podle zadání měl být algoritmus implementován jako nadstavba pro program ArcGIS. Od této implementace jsem však musela odstoupit z několika důvodů. Podle specifikací požadavků pro SDK ArcGISu (ESRI, *ArcGIS Resource Center*) je pro programování v C++, na rozdíl od Javy a .NET, nutný ArcGIS Engine Runtime a komerční verze Microsoft Visual Studia. Komerční verze VS sice poskytují 90-denní trial verzi, ovšem v průběhu této doby jsem neměla k dispozi ArcGIS Engine Runtime, který se pro verzi 10 nepodařilo získat ani po konzultaci s firmou ARCDATA Praha. Na katedře dostupná verze 9.2 zase nemohla být naistalována z důvodu naistalované novější verze ArcGIS Desktopu 10 s roční licencí poskytnutou katedrou.

Tyto problémy s licencemi tak vyústily v nutnost řešit vzniklý problém jinak. Z uživatelského hlediska sice výsledné řešení přináší nutnost spustit samostatný program mimo prostředí ArcGIS, na druhé straně však také pracuje s nativním formátem ESRI (shapefile) a navíc uživatel nemusí mít tento komerční software (včetně nutných extenzí) vůbec naistalovaný. Prohlížet, vybírat a editovat shapefile je možné např. pomocí open source programu *QuantumGIS* (`http://www.qgis.org/`).

V dalších částech jsou nejprve stručně představeny využité knihovny a dále následuje vlastní postup implementace.

7.1 Knihovny CGAL, Boost a ShapeLib

Knihovna CGAL (*Computational Geometry Algorithms Library*) je volně dostupná knihovna napsaná v jazyce C++, která umožňuje přístup k efektivním a spolehlivým algoritmům v oblasti výpočetní geometrie, jako jsou například triangulace, Voronoi teselace nebo tvorba konvexních obálek. Projekt byl založen v roce 1996 sedmi výzkumnými institucemi a od té doby se stále rozšiřuje. V současné době je nejnovější verzí verze 4.0 uvolněná v březnu 2012. Pro zpracování diplomové práce byla použita verze 3.9.

Boost knihovny jsou taktéž psány v jazyce C++. Jejich instalace je nutná už při instalování knihovny CGAL, a pro další práci byla navíc využita knihovna *BGL - The Boost Graph Library*, která implementuje rozličné grafové algoritmy. Nejnovější verze boost knihoven je 1.49.0, uvolněná v únoru 2012, pro zpracování diplomové práce byla využita verze 1.47.0.

Knihovna ShapeLib napsaná v C slouží pro čtení, zápis a editaci shapefilů. Jejím autorem je Frank Warmerdam, první verze je z roku 1998. Dokumentace knihovny je stručná, ale vzhledem k její jednoduchosti dostatečná. Nevýhodou je neexistence veřejného fóra, ale dotazy mohou být posílány přímo autorovi. Nejnovější verze je 1.3.0, diplomová práce byla zpracována pomocí verze 1.2.10.

7.2 Určení budov pro agregaci

Finálním výstupem první fáze algoritmu je dvojrozměrný vektor typu `vector <vector<int>>`, nesoucí v každém řádku seznam ID polygonů, které se mají agregovat. ID řádku je zároveň ID nového polygonu, vzniklého z odpovídajících záznamů. V následujících podkapitolách je rozveden postup, jak k tomuto finálnímu poli dojdeme.

7.2.1 Načtení shapefilů

Jako první je požadováno zadání cest k shapefilům (k samostatným souborům s koncovkou .shp, ne k feature classes uloženým v geodatabázi) se zdrojovými daty. Jeden shapefile by měl obsahovat polygonovou vrstvu budov. Tato vrstva musí splňovat následující podmínky.

- polygony nesmí obsahovat díry,
- orientace polygonů musí být běžná pro shapefile (uzavřený polygon je zadán body po směru hodinových ručiček),
- vrstva musí mít v atributové tabulce sloupec se spočtenou plochou polygonu (s názvem „Shape_Area“).

Druhá vrstva je vrstva liniová obsahující (osy) cestní sítě.

Shapefily jsou načítány pomocí funkcí knihovny ShapeLib. Funkcí SHPOpen se vytvoří objekt typu SHPHandle, který je dále využit pro zjištění informací o shapefilu, především typu geometrie (point, line, polygon, multipoint) a počtu záznamů (viz kód 7.1).

Kód 7.1: Načtení shapefilu

```
SHPHandle shpHandle;
shpHandle = SHPOpen("D:\\Dokumenty\\zkusebni\\zkusebni", "rb");
SHPGetInfo(shpHandle,&pnEntities,&pnShapeType,NULL,NULL);

for (int i=0;i<pnEntities;i++) {

      SHPObject* shpObject = SHPReadObject(shpHandle,i);

            for (int j=shpObject->nVertices-1;j>0;j--) {
            x=CGAL::to_double(shpObject->padfX[j]);
            y=CGAL::to_double(shpObject->padfY[j]);
            Point_2 bod2(x,y);
            poly_shp2[i].push_back (bod2);}

      SHPDestroyObject(shpObject); } }
```

Každý prvek shapefilu je reprezentován objektem typu SHPObject, který je návratovým typem funkce SHPReadObject. Procházením shapefilu se dostáváme ke konkrétním prvkům, jejichž souřadnice lze uložit do vhodné datové struktury. Vzhledem k tomu, že k dalším operacím s polygonem je využita knihovna CGAL, jsou souřadnice ukládány do datových struktur `Point_2` a `2D Polygon` této knihovny. Linie jsou načítány do struktury `Segment_2` (tj. úsečka).

Při načítání jednotlivých prvků shapefilu je kontrolována plocha objektu, neboť algoritmus nebude zpracovávat budovy s plochou menší S_{min} (část 6.2). Hodnota plochy je načítána přímo z atributové tabulky (souboru .dbf) shapefilu. Podobně jako v případě čtení geometrie je nutné použít funkci pro otevření databáze DBFOpen (kód 7.2).

Kód 7.2: Načtení atributů

```
DBFHandle dbfHandle;
dbfHandle = DBFOpen("D:\\Dokumenty\\zkusebni\\zkusebni", "rb");
area=DBFReadDoubleAttribute(dbfHandle,i,DBFGetFieldIndex(dbfHandle,"Shape_Area"));
```

Do proměnné `area` je načítána plocha ze sloupce „Shape_Area“ a je porovnávána s kritickou hodnotou S_{min}. Pokud je plocha menší, geometrie objektu se vůbec nenačte a pokračuje se dalším prvkem.

7.2.2 Voronoi diagram a určení sousednosti polygonů

Další fází je vytvoření Voronoi diagramu nad centroidy načtených polygonů. Voronoi diagramy jsou následně využity pro hledání sousedních polygonů. Centroidy jsou vypočítávány pro každý polygon ihned po jeho načtení dle vztahů 6.1 a 6.2 a uloženy do samostatného vektoru. Zároveň je každému centroidu přiřazen index shodný s ID polygonu.

Voronoi diagram je generován pomocí knihovny CGAL. Algoritmus implementoval Menelaos Karavelas a to ve formě adaptoru 2D Delaunay triangulace. Výsledná teselace je uložena ve struktuře DCEL, která umožňuje standardní operace: procházet hrany buněk, zjišťovat sousední buňky, procházet hrany vycházející z daného vertexu a další. Z uživatelského hlediska jsou diagramy konstruovány pomocí příkazu `insert(generátor buňky)`:

```
vd.insert(centr[i]);
```

Pro nalezení navzájem sousedících polygonů jsou postupně procházeny všechny buňky vytvořené Voronoi teselace. Pomocí funkce `dual()` je přístupný generátor aktuální buňky. Prochází se hrany aktuální buňky a pro každou z nich je určen sousední centroid.

Spojnice centroidů vytvoří úsečku. Testuje se, zda protíná některou z linií cestní sítě postupným testováním, dokud se najde/nenajde průsečík. Pokud neprotínají, vypočítá se minimální vzdálenost mezi příslušnými polygony, viz část 4.2. Pokud je vzdálenost menší než d_{max} je index testovaného centroidu (tj. i polygonu) přidán do spojového seznamu (*linked list*) aktuálního centroidu (polygonu). Tím je vytvořen seznam sousedů pro agregaci pro každý polygon (kontejner `vecRef2`) - dle alg. 6.1, viz kód 7.3.

Kód 7.3: Vytvoření seznamu sousedů

```
for (Face_it fit = vd.faces_begin(); fit != vd.faces_end(); ++fit){
	//index polygonu, ktery je zdrojem face
	int ind = fit->dual()->point().index();
	//prochazeni hran face
	Ccb_halfedge_circulator ec_start = fit->outer_ccb();
	Ccb_halfedge_circulator ec = ec_start;
	do {
		int refer = ec->twin()->face()->dual()->point ().index ();
		//spojnice centroidu aktualniho polygonu a sousedniho
		Segment spojnice(fit->dual()->point(),ec->twin()->face()->dual ()->point());
			//pro kazdou linii v cestni siti
			for (unsignedint q=0;q<seg_shp.size ();q++){
			if (CGAL::do_intersect(spojnice,seg_shp[q])) break;
			elseif (q==seg_shp.size ()-1) {
				double min=CalcMinDistance(poly_shp[ind],poly_shp[refer]);
					if (min<=5) {
					//seznam sousedu pro polygon
					vecRef2[ind].insert_from_back(refer);}}
				}
			}
			while (++ec!=ec_start);
			indO++;}
```

Kód 7.4: Prohledávání do hloubky a naplnění kontejneru ToAggreg

```
//prohledavani do hloubky
for (int u=0;u<vecRef2.size ();u++){
	if (stav[u]==-1){
		DFS(u); calc++;}}

voidDFS(int u) {
	paths[calc].push_back (u);
	stav[u]=0;

	node *temp1;
	temp1=vecRef2[u].head;

	while (temp1 != NULL) {
		if (stav[temp1->data]==-1){
			p[temp1->data]=u;
			//rekurzivni volani DFS
			DFS(temp1->data);
			}
		temp1=temp1->next;}

	stav[u]=1;}

//nasledny seznam nese ID polygonu pro agregaci
for (int i=0;i<paths.size();i++){
	if (paths[i].size()>1)
		ToAggreg.push_back (paths[i]);}
```

Pro nalezení seznamů budov pro agregaci je použit algoritmus prohledávání do hloubky (viz část 6.1). Jak bylo uvedeno v části 4.5.2, výslednými produkty tohoto algoritmu jsou časové značky otevření a uzavření vrcholů a vektor předchůdců. Tyto vektory nejsou v našem případě potřeba, naopak jsou zavedeny dva nové dvojrozměrné vektory `paths` a `ToAggreg`. Oba jsou typu `vector<vector<int>>`. Protože před začátkem prohledávání nevíme, kolik izolovaných podstromů (tj. kolik skupin polygonů pro agregaci) bude v grafu objeveno, je pomocnou strukturou při prohledávání do hloubky vektor `paths`, jehož délka odpovídá počtu polygonů. Po ukončení prohledávání jsou hodnoty vektoru `paths` překopírovány do vektoru `ToAggreg`, aby se eliminovaly prázdné řádky tohoto vektoru. Zároveň pak pořadí ve vektoru `ToAggreg` určuje index nového agregovaného polygonu (kód 7.4).

7.3 Vlastní agregace

7.3.1 Konstrukce straight skeletonu a nalezení minimální kostry grafu

Po naplnění kontejneru `ToAggreg` je pro každý polygon v něm obsažený zkonstruován straight skeleton pomocí funkce knihovny CGAL `create_interior_straight_skeleton_2`. Výsledná skeletonizace je uložena v datové struktuře HDS, která umožňuje podobné operace jako struktura DCEL.

Z vytvořených straight skeletonů jsou vybrány pouze vnitřní vrcholy - získáme redukované straight skeletony. Pro zjištění, zda se jedná o vnitřní uzel skeletonu, je možné využít funkci CGAL `is_skeleton()`. Odpovídající uzly jsou načítány do dvojrozměrného vektoru `vector<vector<PointRec>> pointsOfGraph`. Dále jsou připravena data pro spuštění funkce pro nalezení minimální kostry grafu.

Jsou vytvořeny všechny možné (ve své podstatě orientované) hrany mezi body, jejichž indexy jsou uloženy do kontejneru `vector<vector<int>> Edges` a zároveň jsou vypočítávány jejich váhy a ukládány do vektoru `vector<vector<double>> Eweights`. Naplněné kontejnery představují vstupní argumenty kruskalova algoritmu pro nalezení minimální kostry grafu. Funkce je z knihovny BGL (kód 7.5 a 7.6).

Nalezená kostra grafu, tj. redukovaný skeleton agregovaného polygonu, je uložena ve formě spojové reprezentace grafu do nového kontejneru `vector<vector<vector<int>>> MST`. Tento kontejner je trojrozměrný, protože udržuje informaci o každém novém polygonu, ke každému polygonu udržuje seznam vrcholů a ke každému vrcholu seznam sousedících vrcholů.

Kód 7.5: Naplnění kontejneru bodů, hran a jejich vah pro MST

```
//prochazet kazdy skeleton
for (int s=0;s<skeletons.size();s++){
	for (int s2=0;s2<skeletons[s].size ();s2++)
	{
		//prochazet vertexy skeletonu
		Ss& ss=*skeletons[s][s2];
		for (Vertex_const_iterator i = ss.vertices_begin(); i !=
ss.vertices_end(); ++i)
			{
		const Point_2 p = i->point ();
		//pokud je prochazeny bod typu skeleton vertex, je pridan do vektoru
bodu pro dalsi zpracovani
				if (i->is_skeleton())
					pointsOfGraph[s].push_back (p);
			}
		}
	}
//naplnění kontejneru pro hrany a jejich vahy
for (int q=0;q<2;q++){
	for (int q1=0;q1<pointsOfGraph[q].size ();q1++){
		for (int q2=q1+1;q2<pointsOfGraph[q].size ();q2++){
		E e;
		e.first=q1; e.second=q2;
		Edges[q].push_back (e);
		Eweights[q].push_back
(CalcDistance(pointsOfGraph[q][q1],pointsOfGraph[q][q2]));}
		}
	}
```

Kód 7.6: Nalezení minimální kostry grafu

```
//Vytvoreni grafu pro kazdy vektor ID polygonu pro spojeni
for (unsignedint i=0;i<pointsOfGraph.size ();i++){

	constint num_nodes = pointsOfGraph[i].size ();
	std::size_t num_edges = Edges[i].size ();

	Graph g(num_nodes);
	property_map<Graph, edge_weight_t>::type weightmap = get(edge_weight, g);
	//pridavani hran do grafu z vektoru Edges a prirazeni vah z vektoru Eweights
	for (std::size_t j = 0; j < num_edges; ++j) {
	Edge e; bool inserted;
	boost::tie(e, inserted) = add_edge(Edges[i][j].first ,Edges[i][j].second, g);
	weightmap[e] = Eweights[i][j];
		}
	//nacteni hodnot vah do property_map
	property_map < Graph, edge_weight_t >::type weight = get(edge_weight, g);
	std::vector < Edge > spanning_tree;
	//kruskaluv algoritmus pro hledani minimalni kostry grafu (MST)
	kruskal_minimum_spanning_tree(g, std::back_inserter(spanning_tree)); }
}
```

7.3.2 Generalizace redukovaného skeletonu a tvorba první aproximace polygonu

Podle algoritmu 6.2 jsou na základě informací z kontejneru MST uloženy všechny větve každého polygonu do kontejneru `vector<vector<list<int>>> branches`. Tento kontejner je opět trojrozměrný, protože nese informaci o každém polygonu, ke každému polygonu udržuje seznam větví a každá větev je tvořena posloupností vrcholů (jejich indexů). Struktura seznam (*list*) je nejen pro kontejner `branches` použita kvůli následujícím výhodám:

- seznam umožňuje přidávání prvků na konec i na začátek seznamu,
- seznam umožňuje rychlé smazání všech prvků dané hodnoty.

První výhoda seznamu je využita už při sestavování větví, neboť kvůli správnému pořadí vrcholů je nutné při prohledávání přidávat vrcholy jak na začátek, tak na konec seznamu.

Následuje generalizace větví podle algoritmu 4.6 (Douglas-Peucker) a odstranění krátkých větví. Výstupem algoritmu však není seznam bodů, které mají v dané větvi zůstat. Naopak, výstupem je kontejner `vector<list<int>> toErase`, který nese seznam indexů vrcholů, které se mají smazat. Šířka koridoru h pro Douglas-Peucker algoritmus je určena dle vztahu 6.3.

Dále je vytvořen kontejner `vector<list<int>> finalSkeleton`. V něm je uložen topologicky správný a proti směru hodinových ručiček orientovaný redukovaný skeleton agregovaného polygonu (viz část 6.2.2). Zbývá tyto dva kontejnery, `finalSkeleton` a `toErase`, zkombinovat: ze seznamu `finalSkeleton` jsou smazány všechny indexy vrcholů, které jsou uloženy v seznamu `toErase` - získáme generalizovaný redukovaný skeleton.

Posledním krokem před samotnou rekonstrukcí polygonu je tvorba jeho první aproximace, která vznikne posunutím bodů po bisektorech podle vztahů 6.4 a 6.5, kdy $d = 0,001$ (viz část 6.2.2). Souřadnice bodů jsou načítány z kontejneru `pointsOfGraph` a finální vrcholy jsou uloženy v kontejneru `vector<vector<PointRec>> finalPoints`.

7.3.3 Zpětná rekonstrukce polygonu

Jak bylo uvedeno v části 6.2.3, pro zpětnou rekonstrukci polygonu byla využita datová struktura SLAV Felkela a Obdržálka, která byla jen mírně modifikována. Základní definované třídy a struktury jsou následující:

- třída PointRec: nese souřadnice vrcholů (double x, double y)
- třída RayRec: slouží pro uložení polopřímek - bisektorů a hran
- struktura Vertex: ukládá jednotlivé vrcholy v LAV
- struktura Intersection: slouží pro uložení souřadnic, vzdálenosti a typu detekované události
- třída VertexList: udržuje dvojitý kruhový seznam

Celá datová struktura i definované algoritmy jsou poměrně komplexní a popisovat zde celou funkcionalitu algoritmu by nebylo vhodné:

- za prvé, jedná se z velké části o přejatý kód (z důvodu velké podobnosti funkcionality a vhodnosti datové struktury),
- za druhé, kód je rozsáhlý a dávat do textu jen některé ukázky by vyžadovalo velké množství vysvětlujícího textu. Pro zájemce bude proto patrně nejvhodnější podívat se přímo do zdrojového kódu (viz příloha 1), který je podrobně okomentován. Zde bude uvedena jen základní funkcionalita algoritmu.

Vstupní a výstupní typy argumentů funkce pro zpětnou rekonstrukci polygonu `createOffset` jsou následující:

```
vector<PointRec> createOffset(vector<PointRec> &input, double offsetDistance)
```

Vstupní vektor `input` obsahuje seznam vrcholů první aproximace polygonu v counter-clockwise orientaci. Argument `offsetDistance` představuje ukončovací podmínku w. Navrátovým typem je `vector<PointRec>`.

Při načítání vstupního polygonu jsou body typu `PointRec` ukládány do struktury `VertexList`, která je tvořena jednotlivými vertexy. Každý `Vertex` je inicializován třemi body - první nese souřadnice vlastního vertexu, dalšími dvěma jsou předchozí a následující bod vstupního polygonu (viz kód 7.7). Při načítání je vypočten bisektor a definovány sousedící hrany (atributy `axis`, `leftLine`, `rightLine`). Zapojení do LAV (atributy `prevVertex`, `nextVertex`) je provedeno až po načtení všech bodů polygonu, neboť se jedná o pointery. Atributy `leftVertex` a `rightVertex` jsou ukazatele na vrcholy, z nichž daný vrchol vznikl.

Kód 7.7: Konstruktory struktury Vertex

```
Vertex (const PointRec &p, const PointRec &prev = PointRec (), const PointRec &next
     = PointRec ()): point (p), axis (angleAxis (p, prev, next)), leftLine (p,
     prev), rightLine (p, next), leftVertex (NULL), rightVertex (NULL), nextVertex
     (NULL), prevVertex (NULL), done (false), ID (-1), Idist(0.0), offset(1)

Vertex (const PointRec &p, Vertex &left, Vertex &right);
```

Druhý z uvedených konstruktorů slouží pro vytvoření `Vertexu` na místě původního průsečíku - `Vertex` je inicializován souřadnicemi průsečíku a pointery na vrcholy v LAVu, z nichž nový vrchol vzniká.

Do struktury jsou navíc přidány atributy `double Idist` a `int offset`. Ty slouží pro finální rekonstrukci polygonu - viz část 6.2.3.

Po zpracování všech průsečíků splňujících podmínku 6.6 jsou podle algoritmu 6.5 aktualizovány souřadnice výsledného rekonstruovaného polygonu, viz kód 7.8.

Zápis do shapefilu. Návratovým typem funkce `createOffset` je `vector<PointRec>`, ve kterém jsou uloženy vrcholy rekonstruovaného polygonu v clockwise orientaci (orientace pro uzavřený polygon v shapefilu). Ve SLAV jsou uloženy všechny LAV, tedy včetně děr. Pro zjednodušení zápisu do shapefilu je však navrácen pouze ten LAV, který určuje konturu uzavřeného polygonu.

Pro tvorbu shapefilu a zápis dat je podobně jako při načítání využita knihovna ShapeLib. Do shapefilu je společně s geometrií samotnou uložen atribut `double origArea`, ve kterém je uložena hodnota součtu ploch původních polygonů (kód 7.9).

Kód 7.8: Finální update souřadnic ve SLAV

```
//finalni update souradnic
	VertexList :: iterator it;
	for (it = vl.begin(); it != vl.end(); it++) {
		if (!(*it).done ) {
			double innerAngle;
			double aL, aR;
			double l;
			aL = (*it).leftLine.angle; if (aL < 0) aL=2*PI+aL;
			aR = (*it).rightLine .angle; if (aR < 0) aR=2*PI+aR;
			innerAngle = abs(aL-aR)/2.0;

			if (sin(innerAngle) == 0) l=offsetDistance-(*it).Idist;
			else l = (offsetDistance-(*it).Idist )/ (sin(innerAngle));

			(*it).point.x = (*it).point.x + cos((*it).axis .angle )*l;
			(*it).point.y = (*it).point.y + sin((*it).axis .angle )*l;
		}
	}
```

Kód 7.9: Zápis do shapefilu

```
SHPHandle    hSHP;
DBFHandle    hDBF;
int        nShapeType, Vertices;
SHPObject    *psObject;
nShapeType = 5;
hSHP = SHPCreate( "D:\\test124",nShapeType);
int origArea;
hDBF = DBFCreate("D:\\test124");

origArea =DBFAddField(hDBF,"origArea",FTDouble,8,3);

	for (unsigned int e = 0; e<ToAggreg.size (); e++) {
		Vertices = result[e].size();
		double X[150], Y[150];
		for (int w = 0; w<result[e].size (); w++) {
			X[w] = result[e][w].x;
			Y[w] = result[e][w].y;
		}

		psObject = SHPCreateSimpleObject( nShapeType, Vertices, X, Y, NULL);
		int aRec;
		aRec = SHPWriteObject( hSHP, -1, psObject );
		DBFWriteDoubleAttribute(hDBF,aRec, origArea,aggregArea[e]);
	 }
DBFClose( hDBF );
SHPClose( hSHP );
```

7.4 Vytvořený nástroj

Vytvořený nástroj je ve formě konzolové aplikace pro Windows 32-bit. Uživatelským vstupem jsou cesty k shapefilům polygonové a liniové vrstvy, místo uložení a název nově vytvářeného shapefilu. Dále je požadována hodnota S_{min} a měřítkové číslo M.

Výstupem je shapefile obsahující agregované polygony.

8 DOSAŽENÉ VÝSLEDKY A JEJICH ZHODNOCENÍ

Generalizační algoritmus byl testován na devíti vzorových územích, které reprezentují šest dále uvedených typů zástavby, typických pro české prostředí. Algoritmus byl pro každé území spuštěn ve dvou mírách generalizace - s parametry odpovídající měřítkům 1:25 000 a 1:50 000. Ve stejných mírách generalizace byla vzorová data agregována i nástrojem pro agregaci zabudovaném v SW ArcGIS. Dosažené výsledky jsou vzájemně porovnány. Z kartografického hlediska jsou výsledky zhodnoceny vizuálně. Hodnoceno je především:

- zachování pravoúhlosti budov,
- zachování polohové přesnosti budov,
- celková kartografická přirozenost výsledků.

Z objektivního hlediska je hodnocena změna plochy a posun těžiště agregovaných budov.

Kapitola je rozdělena do čtyř částí - první popisuje testovací data, druhá definuje a charakterizuje vybraná vzorová území, třetí posuzuje dosažené výsledky a čtvrtá obecně hodnotí navržený algoritmus na základě dosažených výsledků.

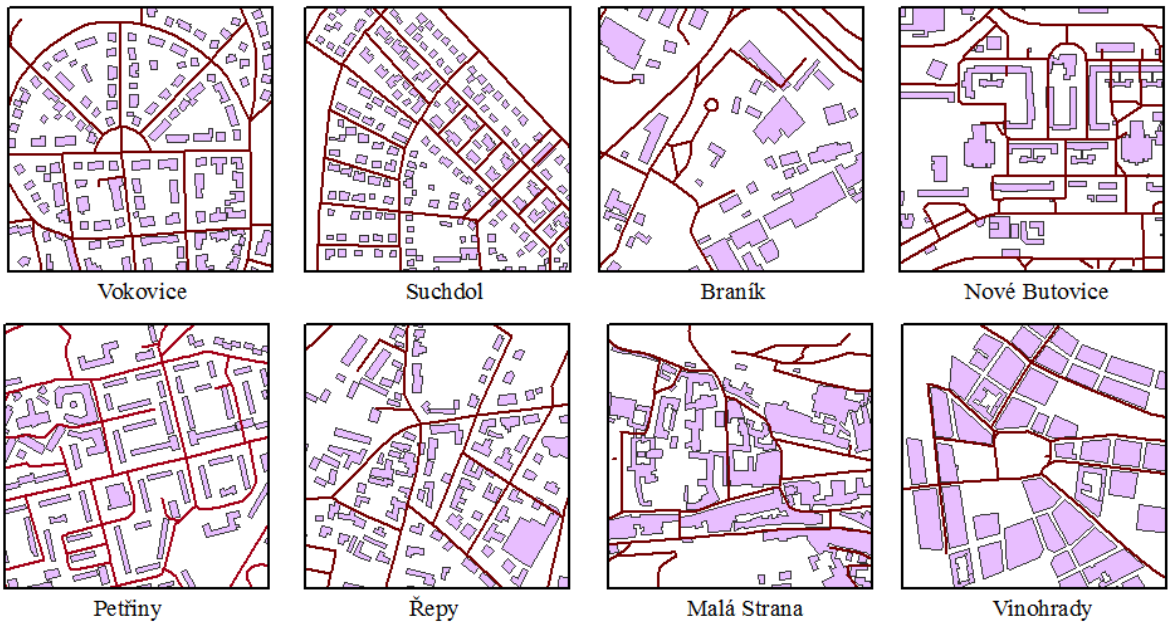
8.1 Testovací data

Algoritmus byl testován na datech ZABAGED a CEDA, data byla k dispozici pro západní část města Prahy a pro účely testování byla poskytnuta katedrou kartografie a aplikované geoinformatiky PřF UK. Z databáze ZABAGED byla použita vrstva 1.02 - budova jednotlivá nebo blok budov, z databáze CEDA byla použita vrstva os cestní sítě pro Prahu. Obě datové vrstvy jsou v měřítku odpovídající podrobnosti 1:10 000.

8.2 Vybraná vzorová území

Pro účely testování bylo definováno šest typů zástavby, typických pro české prostředí, aby mohla být zhodnocena efektivita algoritmu z kartografického hlediska. Pro každý z typů byla vybrána jedna nebo více lokalit, na nichž probíhalo testování. Lokality jsou menšího rozsahu, s počtem vstupních polygonů < 250 (kromě území 2). Uveďme jednotlivé typy zástavby a konkrétní testovací lokality (viz obr. 8.1). V dalším textu je stručně charakterizujme.

- sídliště - sídliště Petřiny, Nové Butovice a Velká Ohrada
- čtvrti rodinných domů a vil - Vokovice, Suchdol
- bloky starších budov - Vinohrady
- historické centrum města - Malá Strana
- centrum menší obce - Řepy
- průmyslový areál - Braník



Obr. 8.1: Výřezy vzorových území (měřítko je proměnlivé v rozsahu 1:10 000 - 1:25 000)

Sídliště. Tento typ zástavby může být poměrně různorodý, a proto je testován na třech různých lokalitách. Může být typický uniformními budovami s pravoúhlým půdorysem a podobnou orientací budov vůči ulici nebo se naopak může vyznačovat různě tvarovanými a vzájemně různě natočenými budovami. Společným rysem však bývá výrazná protáhlost budov.

Čtvrti rodinných domů a vil. Rodinné domy a vily jsou vesměs charakteristické jednoduchým půdorysem ve tvaru čtverce nebo obdélníka, avšak často se u nich vyskytují drobné výstupky. Jako celek jsou typicky zarovnány podél komunikace, čímž vytváří různé vzory - čtverec, obdélník, kruhovou výseč apod. (viz. obr. 8.1). Jedná se o rozvolněnou zástavbu.

Bloky (starších) budov. Jsou typické vyššími, těsně na sebe navazujícími budovami kopírujícími komunikaci, čímž zpravidla vytváří uzavřený blok budov s vnitřním dvorem. V testovacích datech byl tento typ zástavby většinou znázorněn jako blok budov.

Historické centrum města. Je typické hustou zástavbou, různě tvarovanými budovami a užšími ulicemi. Ulice bývají křivolaké a zástavba je kopíruje. Historické budovy mívají složitější půdorys.

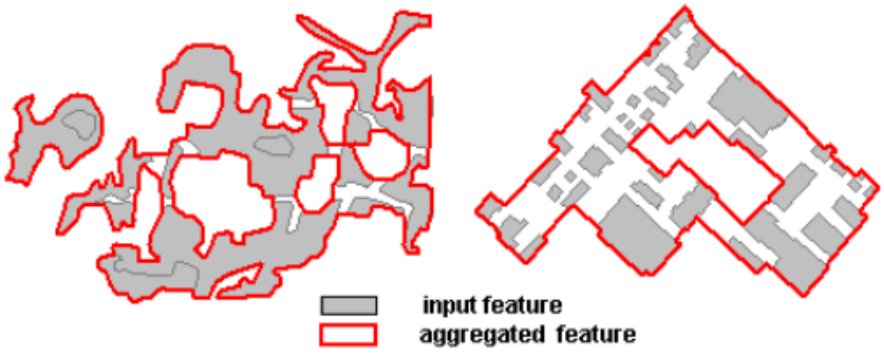
Centrum menší obce. Je typické náměstím (návsí) uprostřed, od kterého se rozebíhají ulice různými směry. Budovy bývají jednoduššího půdorysu (čverec, obdélník), ale vzájemně různě natočené. Zástavba bývá rozvolněnější, ale hustší než v případě čtvrtí rodinných domů.

Průmyslový areál. Podobně jako sídliště může nabývat mnoha podob. Typický je však budovami velkého a malého rozsahu v těsné blízkosti, často různě vzájemně natočenými. Velké budovy mohou mít buď jednoduchý půdorys, nebo naopak půdorys poměrně komplikovaný. Zástavba bývá volnější.

8.3 Testování algoritmu na vzorových územích a porovnání se SW ArcGIS

V této části jsou posouzeny výstupy algoritmu na menších shlucích budov s cílem ohodnotit kartografickou věrnost výsledků. Přílohy 2 - 8 ilustrují generalizaci celých vzorových území (případně většiny jejich rozsahu, v závislosti na jejich velikosti). Tab. 8.1 a 8.2 podávají přehled o změnách ploch a poloze těžiště agregovaných budov. Porovnání se SW ArcGIS je provedeno vizuálně. Funkce ArcGIS pro agregaci je nazvána *Area Aggregate*, uveďme zde její princip.

Area Aggregate. Algoritmus probíhá v rastrovém prostředí. Nejprve je vstupní vektorová vrstva převedena na vrstvu rastrovou. Aplikací funkcí Expand a Shrink (pravděpodobně funkce dilatace a eroze, viz část 3.3.2) dochází k agregaci budov ve specifikované vzdálenosti (uživatelský vstup). Výsledek je převeden zpět na rastr. Základními variantami algoritmu je buď možnost zachování pravoúhlosti (vhodné pro budovy) nebo nikoliv - viz obr. 8.2 (ESRI, 2000).



Obr. 8.2: Ilustrace výsledků funkce Area Aggregate. Agregace neortogonálních prvků (*vlevo*), agregace ortogonálních prvků (*vpravo*). (převzato z: ESRI, 2000, s. 11)

Před vlastním testováním navrženého algoritmu i algoritmu Area Aggregate byla nejprve výběrem zjednodušena vrstva os cestní sítě. Byly vytvořeny dvě nové vrstvy - jedna odpovídající měřítku 1:25 000, druhá odpovídající měřítku 1:50 000 (podkladem pro výběr byly Topografické mapy ČR odpovídajících měřítek). Poté byly spuštěny vlastní algoritmy pro agregaci.

Určení centroidů budov. V tab. 8.1 a 8.2 jsou porovnány vzdálenosti centroidů původních polygonů a výsledného agregovaného polygonu. Souřadnice centroidu výsledného polygonu jsou spočteny dle vztahů 6.1 a 6.2 (část 6.1). Centroid původního shluku budov je určen jako vážený průměr centroidů dílčích budov:

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

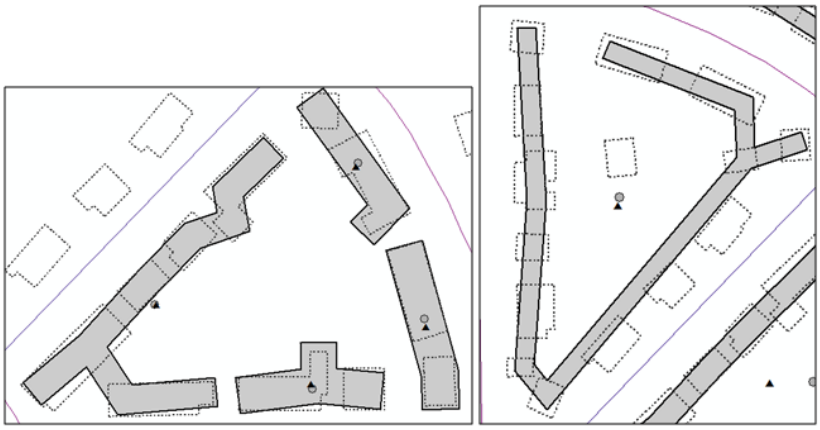
kde w_i je plocha dílčího polygonu.

Před zhodnocením výsledků uveďme legendu platnou pro obrázky 8.3 - 8.9:

Území 1 - Vokovice. Pohled na generalizaci území jako celku přináší příloha 2. Území bylo zvoleno především pro testování, zda je algoritmus schopen zachovat zarovnání budov podél zakřivené komunikace. Z výsledku je zřejmé, že obecně zarovnání nezachová. Příčinou je především generalizace redukovaného skeletonu agregovaného polygonu. Při pohledu na výslednou generalizaci jako celek je nutné konstatovat, že především pro měřítko 1:50 000 je výsledek z kartografického hlediska nepříznivý.

Ze statistického hlediska (tab. 8.1 a 8.2) můžeme konstatovat, že posun těžiště je malý a ve výsledných měřítcích zanedbatelný ($< 2mm$). Obr. 8.3 ilustruje agregaci konkrétních skupin budov. Algoritmus zachová pravé úhly, budovy tvarově zjednodušší. Problematická je agregace většího počtu budov - začínají vznikat nepřirozené útvary (viz i příloha 2). V případě obr. 8.3 můžeme sledovat i větší odchylku od původního umístění budov. Důvody jsou dva:

- generalizace kostry agregovaného polygonu,
- volba ukončovací podmínky w: při agregaci vzdálenějších budov malé plochy dává vztah 6.7 příliš malou hodnotu w.

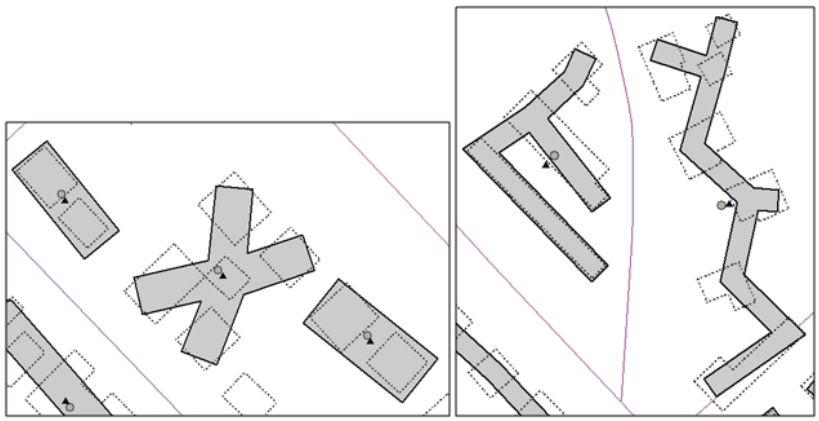


Obr. 8.3: Ilustrace výsledků pro území 1: 1:25 000 (*vlevo*), 1:50 000 (*vpravo*)

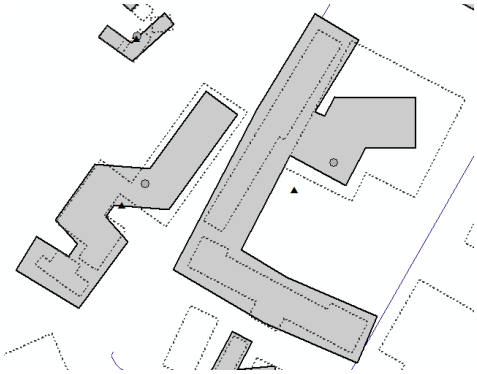
Území 2 - Suchdol. Vzorové území je podobné území 1, ale vzdálenost mezi budovami v rámci bloků jsou menší. Důsledkem je častější (a z kartografického hlediska nepříznivá) agregace budov napříč bloky (viz příloha 3, obr. 8.4 vlevo). Jako celek působí výsledná generalizace obdobně jako v případě území 1 - problematická je generalizace především většího množství budov.

Alogritmus dokáže zachovat pravé úhly, ovšem stejně jako v případě území 1 dochází k větším odchylkám v původním umístění budov.

Území 3 - Braník. Území bylo zvoleno především pro testování, jak se algoritmus vypořádá s agregací blízkých budov různé velikosti a tvaru. Vzhledem k tomu, že při zpětné rekonstrukci dochází k rozšiřování redukovaného skeletonu konstatní rychlostí, nebude pravděpodobně algoritmus schopen správně vystihnout tvar agregovaných budov. Buď budou původně velké budovy příliš malé, nebo naopak původně malé budovy příliš velké. Provedenou agregaci ilustruje obr. 8.5, který potvrzuje tyto domněnky. Je tím také negativně ovlivněna schopnost algoritmu zachovat původní polohu agregovaných polygonů.

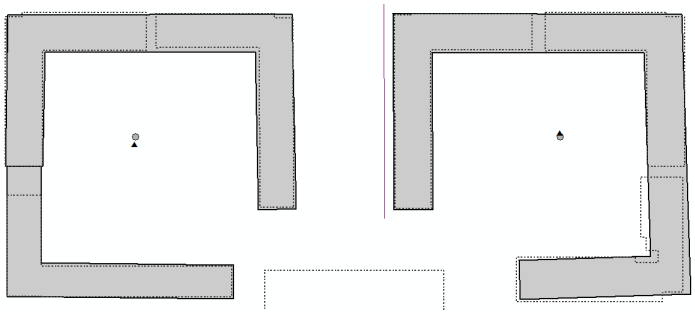


Obr. 8.4: Ilustrace výsledků pro území 2: 1:25 000 (*vlevo*), 1:50 000 (*vpravo*)



Obr. 8.5: Ilustrace výsledků pro území 3: 1:25 000

Území 4 a 5 - Nové Butovice, Velká Ohrada. Výslednou generalizaci území 5 přináší příloha 5. V případě obou území dochází ke generalizaci především pravoúhlých budov srovnatelné šířky. Jejich pravoúhlost i polohová přesnost jsou dobře zachovány (viz i obr. 8.6). V případě generalizace do měřítka 1:50 000 však opět sledujeme výše zmíněné problémy - nepřirozenou agregaci většího množství budov a budov výrazně odlišného tvaru a velikosti v těsné blízkosti.



Obr. 8.6: Ilustrace výsledků pro území 5: 1:50 000

Tab. 8.1: Statistické charakteristiky testování v měřítku 1:25 000

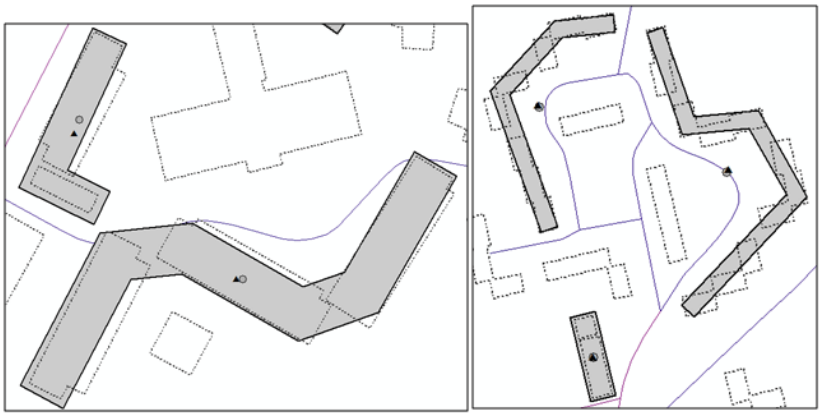
1:25 000	počet budov (vstup)	počet budov (výstup)	počet agreg. budov	průměrná vzdálenost centroidů [m]	SD vzdálenosti centroidů [m]	průměrná změna plochy [m²]	SD změny plochy [m²]
území 1	196	142	24	2,77	1,59	74,34	47,21
území 2	470	329	71	1,95	1,23	85,23	59,63
území 3	111	78	20	5,71	5,91	172,21	204,65
území 4	91	82	6	5,96	5,56	724,75	1088,24
území 5	45	32	8	2,14	0,95	147,69	43,54
území 6	153	138	12	2,91	2,14	148,92	140,35
území 7	137	77	26	2,69	2,33	73,05	98,25
území 8	172	107	31	8,20	4,60	616,11	1045,45
území 9	99	84	10	14,29	14,17	1543,67	1857,36
celkem	1474	1069	208	4,20	5,38	264,41	704,22

Tab. 8.2: Statistické charakteristiky testování v měřítku 1:50 000

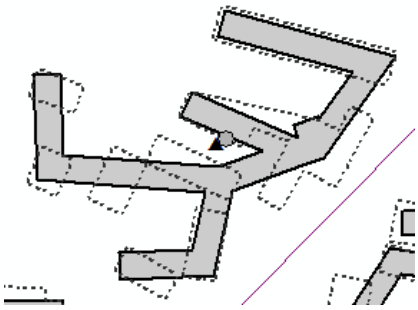
1:50 000	počet budov (vstup)	počet budov (výstup)	počet agreg. budov	průměrná vzdálenost centroidů [m]	SD vzdálenosti centroidů [m]	průměrná změna plochy [m²]	SD změny plochy [m²]
území 1	196	51	26	5,51	6,78	124,39	181,50
území 2	470	145	69	3,36	2,17	79,50	79,29
území 3	111	55	17	7,35	5,57	245,45	407,52
území 4	91	66	16	8,92	5,18	283,82	367,64
území 5	45	23	10	6,20	9,24	393,62	705,76
území 6	153	96	25	3,84	2,98	105,66	69,92
území 7	137	36	18	4,97	3,26	175,44	282,67
území 8	172	86	20	12,62	8,79	1836,89	2156,12
území 9	99	60	16	16,57	12,78	2638,69	2854,40
celkem	1474	618	217	6,47	7,22	488,85	1303,35

Území 6 - Petřiny. Příloha 6 dokumentuje výslednou generalizaci. Na území je testováno, jak algoritmus zpracuje budovy vzájemně různě natočené, které mají srovnatelnou šířku. Generalizaci ilustruje i obr. 8.7. Výsledky jsou v tomto případě poměrně uspokojivé. Agregované polygony zachovávají polohu původních budov a budovy jsou dobře zjednodušené.

Území 7 - Řepy. Území bylo zvoleno pro posouzení agregace blízkých budov různé velikosti, které jsou vzájemně různě natočené. Vzhledem k blízkosti jednotlivých budov se jich agreguje poměrně velké množství (obr. 8.8, příloha 7). Ačkoliv posun těžiště výsledných a původních budov není příliš velký, z kartografického hlediska jsou výsledky krajně nepříznivé.



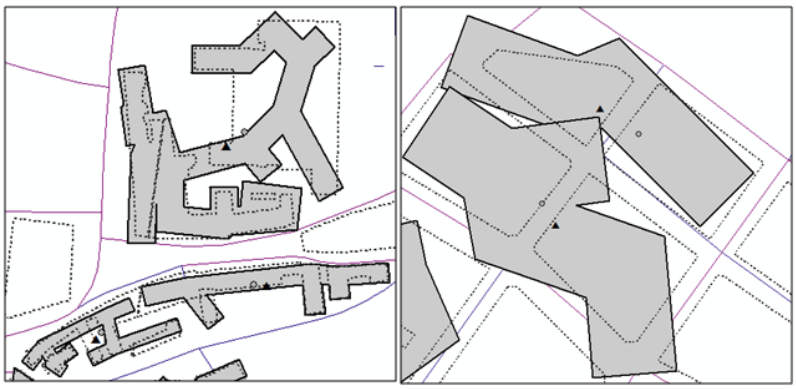
Obr. 8.7: Ilustrace výsledků pro území 6: 1:25 000 (*vlevo*), 1:50 000 (*vpravo*)



Obr. 8.8: Ilustrace výsledků pro území 7: 1:50 000

Území 8 - Malá Strana. Území obsahuje velké množství různorodých budov. Budovy jsou vzájemně různě natočené, v těsné blízkosti se nacházejí budovy různých tvarů. Výsledek generalizace dokumentuje příloha 8. Jako celek působí výsledek relativně dobře v porovnání s výsledkem generalizace v SW ArcGIS. Při bližším zkoumání jednotlivých agregovaných budov však opět objevujeme dříve uvedené skutečnosti (obr. 8.9 vlevo): problematická agregace budov různé šířky, problematická agregace velkého množství budov, ale i poměrně zdařilá agregace budov podobné šířky.

Území 9 - Vinohrady. Zástavba území je ve vzorových datech znázorněna jako bloky budov. Je tedy testována schopnost algoritmu agregovat bloky budov. Musíme konstatovat, že výsledky jsou velmi nevhodné, ať už se jedná o hledisko kartografické (obr. 8.9 vpravo) nebo statistické (velký posun centroidů). Navíc dochází k překryvům sousedících agregovaných polygonů. Problém překryvů algoritmus vůbec neřeší, k překryvům proto může dojít i v jiných případech (i když nebyly při testování zaznamenány).



Obr. 8.9: Ilustrace výsledků pro území 8: 1:50 000 (*vlevo*), ilustrace výsledků pro území 9: 1:50 000 (*vpravo*)

Porovnání se SW ArcGIS. Ukázky agregace vzorových území funkcí Area Aggregate ilustruje obr. 8.10. Algoritmus agreguje budovy vyplněním prázdného prostoru mezi nimi. Dobře zachovává pravé úhly. Z hlediska kartografické přirozenosti poskytuje obecně lepší výsledky než navržený algoritmus - především skutečně dokáže dobře zpracovat široké spektrum budov i v souvislosti s okolními budovami.

Oproti navrženému algoritmu provádí agregaci bez ohledu na omezení liniovými prvky. Agregované budovy nejsou tvarově zjednodušeny.

Jako celek lze výsledky dosahované navrženým algoritmu porovnat s algoritmem Area Aggregate na základě příloh 2-8. Agregace v SW ArcGIS působí ve většině případů přirozeněji - na rozdíl od navrženého algoritmu totiž dokáže budovy agregovat tak, že vytvoří vizuálně pěkné bloky budov (viz. např. obr. 8.10 dole). Navržený algoritmus se v těchto případech potýká s tím, že vytváří nepřirozené polygony (např. obr. 8.8), což má dle mého názoru největší dopad na negativní působení výsledné agregace na vzorových územích.

Můžeme však objevit i některé nedostatky algoritmu Area Aggregate. Protože algoritmus vůbec nezjednodušuje tvar agregovaných budov, lze pozorovat např. vznik velmi úzkých propojení mezi dvěma agregovanými budovami nebo budovy s velkým množstvím malých výstupků.

Ač ani Area Aggregate algoritmus není dokonalý, musíme na závěr zkonstatovat, že v současné chvíli podává lepší výsledky než navržený algoritmus.

Obr. 8.10: Generalizace vybraných budov generalizačním algoritmem *Area Aggregate*

8.4 Zhodnocení navrženého algoritmu a jeho implementace

V této podkapitole bude diskutována kvalita dosažených výsledků a efektivita implementace algoritmu. Návrh algoritmu byl podrobně popsán v 6. kapitole. Byly v ní naznačeny některé potenciální problémy algoritmu. Shrňme nyní ve třech bodech skutečnosti, které algoritmus v současné podobě omezují:

1. kvalita generalizace redukovaného straight skeletonu agregovaného polygonu,
2. zpětné rozšiřování polygonu probíhající konstantní rychlostí pro všechny vrcholy,
3. volba ukončovací podmínky w.

Zhodnocení dosažených výsledků. Dosažené výsledky jsou ovlivněny výše uvedenými skutečnostmi. Zhodnoťme nejprve úspěšnost agregace z pohledu agregace několika budov mimo geografický kontext.

Algoritmus dává dobré výsledky při agregaci menšího množství jednoduchých pravoúhlých budov i budov obecnějších tvarů (viz např. obr. 8.3 vlevo, obr. 8.6, obr. 8.7).

V případě agregace více menších budov, zarovnaných v řadě a s většími rozestupy (např. obr. 8.3 vpravo), vnikají příliš úzké polygony. Příčinou je volba ukončovací podmínky w (vztah 6.7). Pro tyto případy, kdy:

- počet agregovaných budov je relativně velký (> 5),
- poměr plochy agregovaných budov a délky kostry výsledného polygonu je příliš malý (např. na hraně grafických limitů),

by bylo vhodné upravit podmínku w. Úprava by mohla spočívat ve změně koeficientu vztahu 6.7, např.

$$w = \frac{\sum_{i=0}^{n} S_i}{\sum_{j=0}^{m} l_j} \cdot \frac{1}{4}$$

nový koeficient by bylo nutné dát do souvislosti s konkrétním případem, což by vyžadovalo další testování těchto případů.

Třetím diskutovaným případem je problém agregace dvou budov rozdílné šířky (např. obr. 8.5). Tento problém je z logiky návrhu algoritmu vpodstatě neřešitelný, jedná se o důsledek výše zmíněných omezení (bod 2). Potenciálním možným řešením by mohlo být udržení informace o (např. průměrné) vzdálenosti vrcholů každého polygonu od jeho redukovaného straight skeletonu (označme tento atribut jako *vzdál*). Atribut *vzdál* by byl využit při tvorbě první aproximace agregovaného polygonu. Délka posunutí jednotlivých vrcholů kostry by byla přímo úměrná atributu *vzdál* každého z vrcholů. Vlastní rekonstrukce polygonu by již probíhala konstantní rychlostí. Zda by tento přístup přinášel lepší výsledky by opět vyžadovalo další testování.

Zmiňme ještě případ agregace dvou bloků budov (obr. 8.9 vpravo). Příčinu chování algoritmu se v tomto případě nepodařilo z teoretického hlediska objasnit. Příčinou může být chyba v implementaci (pravděpodobné slabé stránky implementace budou ještě diskutovány), pro tento případ se však nepodařila objasnit.

Na závěr zhodnoťme celkové působení provedené generalizace na vzorových územích (přílohy 2-8). Bohužel je nutno konstatovat, že z kartografického pohledu generalizace jako celek příliš dobře nepůsobí. Důvody jsou především následující:

- výskyt úzkých polygonů, tento problém by mohl být odstraněn metodou uvedenou výše,
- v některých místech nezachování linie podél komunikace (problematika generalizace, viz část 6.2.1),
- vznik nepřirozených tvarů při agregaci velkého množství budov (např. obr. 8.8).

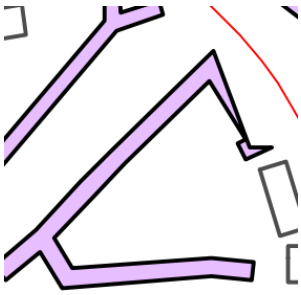
Pokud by se podařilo odstranit jednotlivé dílčí problémy agregace, pravděpodobně by výsledky byly výrazně kartograficky příznivější. Vyžadovalo by to však ještě nemalý objem práce.

Můžeme konstatovat, že algoritmus je schopný zpracovat budovy složitějších tvarů (kromě budov s dírami, viz dále), ovšem dosažené výsledky jsou ovlivněny především polohou, tvarem a orientací sousedících budov, které se liší pro každý konkrétní případ.

Omezení algoritmu. Mimo výše uvedené skutečnosti algoritmus nedokáže zpracovat budovy s dírami (tj. například budovy s vnitřním dvorem). Důvodem je především složitost zpracování. Polygonům s dírami by bylo nutné věnovat velkou pozornost nejen při tvrobě první aproximace polygonu, ale především při zpětné rekonstrukci polygonu. Oba problémy jsou značně složité a dosáhnout úspěšné implementace by bylo velmi náročné (alespoň pro studenta neiformatického oboru).

Zhodnocení implementace algoritmu. Algoritmus se podařilo víceméně úspěšně implementovat v jazyce C++ a vytvořit konzolovou aplikaci pro Windows. Zatím byla objevena jedna slabé stránka implementace, a to pravděpodobně v místě konstrukce první aproximace polygonu nebo v místě generalizace kostry. U některých výsledných polygonů (např. příloha 2, viz obr. 8.11) můžeme sledovat „ulítlý“ vrchol, tedy vrchol, který při konstantním rozšiřováním první aproximace polygonu nikdy nemůže vzniknout. Přesné místo chyby se ale zatím nepodařilo objevit.

Z hlediska časové složitosti je nejnáročnější testování průsečíků s omezeními (alg. 6.1), výpočet vzdálenosti dvou polygonů (část 4.2) a konstrukce straight skeletonu (část 4.3). Délka běhu algoritmu je závislá především na počtu budov, které jsou určené pro agregaci. Při počtu budov určených pro agregaci do 1000 probíhal algoritmus do 20 sekund, při 1200 agregovaných budovách 25 sekund, při 2500 budovách 51 sekund. Při testování většího množství budov bohužel došlo k pádu aplikace (v průběhu zpracování). Při zpracování cca 28 000 vstupních budov by odhadem algoritmus probíhal déle než 10 minut a při dalším navyšování počtu budov by pravděpodobně doba zpracování narůstala strměji.



Obr. 8.11: Chyba implementace

Zhodnocení vhodnosti použitých metod. Využití straight skeletonu a grafových algoritmů pro agregaci budov dosud nebylo v literatuře publikováno (nebo se v záplavě odborných článků nepodařilo objevit). Jedná se tedy ve své podstatě o průkopnickou práci v oblasti agregačních algoritmů. Samotnou myšlenku tvorby topologické kostry, jejího převedení na redukovaný straight skeleton a následnou agregaci koster aplikací algoritmu pro nalezení minimální kostry grafu považuji za úspěšnou. Její realizace je však dosti složitá. Navíc pouhou aplikací tohoto postupu nemůžeme dosáhnout optimálních výsledků pro všechny (nebo alespoň většinu) vstupní konfigurace budov. Jako největší problém (navíc těžko řešitelný) bych uvedla již diskutovanou neschopnost algoritmu agregovat větší množství budov do kartograficky dobře vypadajících bloků budov.

Nicméně ještě jednou zdůrazněme, že se jedná o zcela nový přístup k agregaci budov a bylo by naivní se domnívat, že hned prvním pokusem dosáhneme optimálních výsledků pro každou situaci. Navržený algoritmus skrývá potenciál, bylo by však nutno jej ještě přepracovat a doplnit. Jako negativní faktor se navíc jeví poměrně velká časová složitost algoritmu.

Závěrečné shrnutí výsledků. Algoritmus prokázal dobrou schopnost agregovat menší množství budov při současném zjednodušení agregovaných polygonů. Dosažené výsledky jsou kartograficky přirozené, zachovávají pravé úhly i polohovou přesnost původních budov.

Slabšími stránkami algoritmu je vznik příliš úzkých polygonů v případě agregace většího množství budov, tento problém by měl být odstranitelný změnou podmínky w. Dále je problematická generalizace kostry agregovaného polygonu, což se projevuje např. v nedodržení linie komunikace. Tento bod patří zajisté k jedněm z možností dalšího rozpracování práce. Třetím problémem je vznik nepřirozených, dlouhých polygonů v případě agregace velkého množství budov, kdy by z kartografického hlediska měl agregovaný polygon tvořit uzavřený blok budov.

Do budoucna by bylo možné práci ještě rozšířit o zpracování budov s dírami, což, jak bylo uvedeno výše, je poměrně složitý problém.

9 ZÁVĚR

Předložená diplomová práce se zabývá tématem automatizované kartografické generalizace. Jejím hlavním cílem bylo navrhnout nový generalizační algoritmus pro agregaci budov.

V první, teoretické části diplomové práce, byl podán přehled dosud publikovaných algoritmů používaných pro agregaci budov. Byly zmíněny algoritmy pracující ve vektorovém i rastrovém prostředí. Další kapitoly se již věnovaly návrhu nového algoritmu. Z formálního hlediska byly popsány použité pomocné datové struktury a algoritmy, byly stanoveny kartografické a geometrické podmínky pro agregaci. Vlastní algoritmus je založen především na myšlence redukce polygonů na topologickou kostru metodou straight skeletonu, ze které je získán redukovaný straight skeleton. Agregací redukovaných straight skeletonů je vpodstatě provedena i vlastní agregace budov. Výsledný polygon je zrekonstruován postupným rozšiřováním topologické kostry pohybem po bisektorech směrem ven.

Výstupem praktické části diplomové práce je konzolová aplikace implementovaná v jazyce C++. Aplikace byla otestována na datových sadách ZABAGED a CEDA. Pro účely testování bylo vybráno několik vzorových území, na kterých byl algoritmus spuštěn ve dvou variantách vstupních parametrů. Dosažené výsledky byly zhodnoceny a stručně porovnány s výsledky dosahovanými nástrojem implementovaným v SW ArcGIS.

V úplném závěru práce byly diskutovány problémy navrženého algoritmu a naznačena možná řešení těchto problémů. Byla také zhodnocena úspěšnost implementace algoritmu.

SEZNAM POUŽITÝCH ZDROJŮ

AGENT, DA2: *Constraint Analysis* [on-line]. [cit. 5. 3. 2012] Agent Consortium, 8. 2. 2001. Dostupné na WWW: <http://agent.ign.fr/deliverable/DA2.html>

AGENT, DD2: *Selection of Basic Algorithms* [on-line]. [cit. 14. 9. 2011] Agent Consortium, 8. 2. 2001. Dostupné na WWW: <http://agent.ign.fr/deliverable/DD2.html>

AGENT, DD3: *Strategic Algorithms Using Organisations* [on-line]. [cit. 5. 3. 2012] Agent Consortium, 8. 2. 2001. Dostupné na WWW: <http://agent.ign.fr/deliverable/DD3.html>

AICHHOLZER, O. et al.: *A novel type of skeleton for polygons*. The Journal of Universal Computer Science, 1995. pp. 752-761.

AICHHOLZER, O., AURENHAMMER, F: *Straight Skeletons for General Polygonal Figures in the Plane*. Proceedings of the Second Annual International Conference on Computing and Combinatorics, June 17-19, 1996. pp. 117-126.

BASARANER, M., SELCUK, M.: *An Attempt to Automated Generalization of Buildings and Settlement Areas in Topographic Maps*. ISPRS, 2004, pp. 188-193.

BAYER, T.: *Algoritmy v digitální kartografii*. 1. vyd. Karolinum, Praha, 2008.

BAYER, T.: *Automated building simplification using a recursive approach*. In: ICA Symposium on Cartography for Central and Eastern Europe, LNG&C, pp. 121-145, Vienna, Springer, 2009.

BERG, M. de et al.: *Computational Geometry: Algorithms and Applications*. 3. vyd. Springer, Berlin, 2008. 386 s.

CACCIOLA, F.: *A CGAL implementation of the Straight Skeleton of a Simple 2D Polygon with Holes* [on-line]. [cit. 10. 4. 2012]. Dostupné na WWW: <http://www.cgal.org/UserWorkshop/2004/straight_skeleton.pdf>

CGAL: *Computational Geometry Algorithms Library* [on-line]. [cit. 10. 4. 2012]. Dostupné na WWW: <http://www.cgal.org/>

Cplusplus.com [on-line]. [cit. 23. 5. 2012]. Dostupné na WWW: <http://www.cplusplus.com/>

DEMEL, J.: *Grafy a jejich aplikace*. 1. vyd. Academia, Praha, 2002. ISBN 80-200-0990-6.

DUCHENE, C.: *The CartACom model: a generalisation model for taking relational constraints into account*. In: ICA Workshop on Generalisation and Multiple representations, Leicester, 2004.

DUCHENE, C., GAFFURI, J.: *Combining three multi-agent based generalisation models: AGENT, CartACom and GAEL*. In: 13th International Symposium on Spatial Data Handling, Montpellier, France, 2008.

EČER, P.: *Využití Straight skeletonu pro rekonstrukci tvaru střechy z dat laserového skenování*. Diplomová práce, katedra aplikované geoinformatiky a kartografie PřF UK, 2010. Vedoucí práce: Ing. Tomáš Bayer, Ph.D.

EPPSTEIN, D., ERICKSON, J.: *Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions*. Discrete & Computational Geometry 22, pp. 569-592 [on-line]. [cit. 10. 4. 2012]. Dostupné na WWW: <http://www.dma.fi.upm.es/mabellanas/tfcs/skeleton/html/documentacion/Raising%20Roofs,%20Crashing%20Cycles,%20and%20Playing%20Pool.pdf>

ESRI: *Map Generalization in GIS: Practical Solutions with Workstation ArcInfo Software*. An ESRI Technical Paper, 2000 [on-line]. [cit. 1.9.2011]. Dostupné na WWW: <http://support.esri.com/en/knowledgebase/whitepapers/view/productid/43/metaid/297>

ESRI: *ArcGIS Desktop 9.3 Help* [on-line]. [cit. 1.9.2011]. Dostupné na WWW: <http://webhelp.esri.com/arcgisdesktop/9.3>

ESRI: *ArcGIS Resource Center* [on-line]. [cit. 1.9.2011]. Dostupné na WWW: <http://resources.arcgis.com/>

FELKEL, P., OBDRŽÁLEK, Š.: *Straight Skeleton Implementation*. 14th Spring Conference on Computer Graphics, Budmerice, Slovakia, 1998, pp. 210-218.

HAUNERT, J-H.: *Aggregationin Map Generalization by Combinatorial Optimization*. Disertační práce, Leibniz Universität Hannover, 2008. In:Reihe C, Heft 626 der Veröffentlichungen der Deutsche Geodätische Kommission.

HAUNERT, J-H., SESTER, M.: *Using the Straight Skeleton for Generalisation in a Multiple Representation Environment* [on-line]. [cit. 16.4.2012]. In: ICA Workshop on Generalisation and Multiple representations, Leicester, 2004. Dostupné na WWW: <http://ica.ign.fr/Leicester/paper/Haunert-v2-ICAWorkshop.pdf>

HAUNERT, J.-H., SESTER, M.: *Area Collapse and Road Centerlines based on Straight Skeletons*. Geoinformatica, 2008, Vol. 12, No. 2, pp. 169-191.

JÍROVSKÝ, L.: *Teorie grafů* [on-line]. [cit. 13.4.2012]. Diplomová práce, MFF UK, 2010. Dostupné na WWW: <http://teorie-grafu.cz/>

KOLÁŘ, J.: *Teoretická informatika*. 2. vyd. Česká informatická společnost, Praha, 2004.

LAMY S. et al.: *AGENT Project: Automated Generalisation New Technology*. [on-line]. [cit. 14. 9. 2011]. Dostupné na WWW: <http://agent.ign.fr/public/stresa.pdf>

LI, Z.: *Algorithmic Foundation of Multi-Scale Spatial Representation*. CRC Press, 2007. 282 s.

LI, Z. et al.: *Automated Building Generalization Based on Urban Morphology and Gestalt Theory*. International Journal of Geographical Information Science, 2004, Vol. 18, No. 5, pp. 513-534.

MACKANESS, W., RUAS, A., SARJAKOSKI, L. T. (eds.): *Generalisation of Geograhic Information: Cartograhic Modelling and Applications*. International Cartographic Association, 2007. 282 s.

OKABE, A. et al: *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. 2. vyd. Wiley Series in Probability and Statistics, 2000. 671 s.

POPELÍNSKÝ, J.: *Automatizovaná kartografická generalizace říčních sítí*. Diplomová práce, Geografický ústav PřF MU Brno, 2011. Vedoucí práce: Mgr. Karel Staněk, Ph.D.

SESTER, M.: *Generalization Based on Least Squares Adjustment*. Internationl Archives of Photogrammetry and Remote Sensing. Vol. 33, Part B4, Amsterdam, 2000, pp. 931-938.

Shapefile C Library [on-line]. [cit. 23. 5. 2012]. Dostupné na WWW: <http://shapelib.maptools.org/>

SHEA, K. S., McMASTER, R. B.: *Cartographic Generalization in a Digital Environment: When and How to Generalize*. AUTO-CARTO 9, Ninth International Symposium on Computer-Assisted Cartography, Baltimore,Maryland, April 2-7 1989, pp. 56-67.

STEINIGER, S., TAILLANDIER, P.: *Improving Map Generalisation of Buildings by Introduction of Urban Context Rules.* [on-line]. [cit. 7. 3. 2012]. Dostupné na WWW: <http://ncg.nuim.ie/geocomputation/sessions/1C/1C5.pdf>

SU, B. et al..: *Algebraic models for the aggregation of areafeatures based upon morphological operators.* International Journal of Geographical Information Science, 1997, Vol. 11, No. 3, pp 233 - 246.

ŠÍP, M.: *Topologická kostra polygonu a její využití při kartografické generalizaci.* Diplomová práce,katedra aplikované geoinformatiky a kartografie PřF UK, 2007. Vedoucí práce: Ing. Tomáš Bayer, Ph.D.

ŠTAMPACH, R.: *Automatická kartografická generalizace stavebních objektů z KM do map středních měřítek.* Diplomová práce, Geografický ústav PřF MU Brno, 2006. Vedoucí práce: Mgr. Karel Staněk, Ph.D.

The Boost Grahp Library (BGL) [on-line]. [cit. 23. 5. 2012]. Boost C++ Libraries, 2000-2001. Dostupné na WWW: <http://www.boost.org/doc/libs/1_49_0/libs/graph/doc/index.html>

QI, H., LI, Z.: *An Approach to Building Grouping Based on Hierarchical Constraints.* The Internationl Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Vol. 37, Part B2. Peking, 2008.

SEZNAM PŘÍLOH